A Robust and Fast Action Selection Mechanism for Planning*

Blai Bonet **Gábor Loerincs** **Héctor Geffner**

Departamento de Computación
Universidad Simón Bolívar
Aptdo. 89000, Caracas 1080-A, Venezuela
{bonet,gloerinc,hector}@usb.ve

Abstract

The ability to plan and react in dynamic environments is central to intelligent behavior yet few algorithms have managed to combine fast planning with a robust execution. In this paper we develop one such algorithm by looking at planning as real time search. For that we develop a variation of Korf's Learning Real Time A* algorithm together with a suitable heuristic function. The resulting algorithm interleaves lookahead with execution and never builds a plan. It is an *action selection mechanism* that decides at each time point what to do next. Yet it solves hard planning problems faster than any domain independent planning algorithm known to us, including the powerful SAT planner recently introduced by Kautz and Selman. It also works in the presence of perturbations and noise, and can be given a fixed time window to operate. We illustrate each of these features by running the algorithm on a number of benchmark problems.[1]

Introduction

The ability to plan and react in dynamic environments is central to intelligent behavior yet few algorithms have managed to combine fast planning with a robust execution. On the one hand, there is a planning tradition in AI in which agents plan but do not interact with the world (e.g., (Fikes & Nilsson 1971), (Chapman 1987), (McAllester & Rosenblitt 1991)), on the other, there is a more recent situated action tradition in which agents interact with the world but do not plan (e.g., (Brooks 1987), (Agre & Chapman 1990), (Tyrrell 1992)). In the middle, a number of recent proposals extend the language of plans to include sensing operations and contingent execution (e.g. (Etzioni *et al.* 1992)) yet only few combine the benefits of looking ahead into the future with a continuous ability to exploit opportunities and recover from failures (e.g, (Nilsson 1994; Maes 1990))

In this paper we develop one such algorithm. It is based on looking at planning as a real time heuristic search problem like chess, where agents explore a limited search horizon and move in constant time (Korf 1990). The proposed algorithm, called ASP, is a variation of Korf's Learning Real Time A* (Korf 1990) that uses a new heuristic function specifically tailored for planning problems.

The algorithm ASP interleaves search and execution but actually never builds a plan. It is an *action selection mechanism* in the style of (Maes 1990) and (Tyrrell 1992) that decides at each time point what to do next. Yet it solves hard planning problems faster than any domain independent planning algorithm known to us, including the powerful SAT planner (SATPLAN) recently introduced by Kautz and Selman in (1996). ASP also works in the presence of noise and perturbations and can be given a fixed time window to operate. We illustrate each of these features by running the algorithm on a number of benchmark problems.

The paper is organized as follows. We start with a preview of the experimental results, discuss why we think planning as state space search makes sense computationally, and then introduce a simple heuristic function specifically tailored for the task of planning. We then evaluate the performance of Korf's LRTA* with this heuristic and introduce a variation of LRTA* whose performance approaches the performance of the most powerful planners. We then focus on issues of representation, report results on the sensitivity of ASP to different time windows and perturbations, and end with a summary of the main results and topics for future work.

Preview of Results

In our experiments we focused on the domains used by Kautz and Selman (1996): the "rocket" domain (Blum & Furst 1995), the "logistics" domain (Veloso 1992), and the "blocks world" domain. Blum's and Furst's GRAPHPLAN outperforms PRODIGY (Carbonell *et al.* 1992) and UCPOP (Penberthy & Weld 1992) on the rocket domains, while SATPLAN outperforms GRAPHPLAN in all domains by at least an order of magnitude.

Table 1 compares the performance of the new algorithm ASP (using functional encodings) against both GRAPHPLAN and SATPLAN (using direct encodings) over some of the hardest planning problems that we consider in the paper.[2] SATPLAN performs very well

[1] This paper is a slightly revised version of the paper with the same title to appear in the AAAI-97 Proceedings.

[2] All algorithms are implemented in C and run on an

problem	GRAPH steps	GRAPH time	SAT time	ASP steps	ASP time
rocket_ext.a	34	268	0.1	28	6
logistics.b	47	2,538	6.0	51	29
bw_large.c	14	—	524	18	14
bw_large.d	18	—	4,220	25	51
bw_large.e	—	—	—	36	417

Table 1: Preview of experimental results. Time in seconds. A long dash (—) indicates that we were unable to complete the experiment due to time (more than 10 hours) or memory limitations.

on the first problems but has trouble scaling up with the hardest block problems.[3] ASP, on the other hand, performs reasonably well on the first two problems and does best on the hardest problems.

The columns named 'Steps' report the total number of steps involved in the solutions found. SATPLAN and GRAPHPLAN find optimal *parallel* plans (Kautz & Selman 1996) but such plans are not always optimal in the total number of steps. Indeed, ASP finds shorter sequential plans in the first two problems. On the other hand, the solutions found by ASP in the last three problems are inferior to SATPLAN's. In general ASP does not guarantee optimal or close to optimal solutions, yet on the domain in which ASP has been tested, the quality of the solutions has been reasonable.

Planning as Search

Planning problems are search problems (Newell & Simon 1972): there is an initial state, there are operators mapping states to successor states, and there are goal states to be reached. Yet planning is almost never formulated in this way in either textbooks or research.[4] The reasons appear to be two: the specific nature of planning problems, that calls for decomposition, and the absence of good heuristic functions. Actually, since most work to date has focused on divide-and-conquer strategies for planning with little attention being paid to heuristic search strategies, it makes sense to ask: has decomposition been such a powerful search device for planning? How does it compare with the use of heuristic functions?

These questions do not admit precise answers yet a few numbers are illustrative. For example, domain independent planners based on divide-and-conquer strategies can deal today with blocks world problems of up to 10 blocks approximately.[5] That means 10^7 different states.[6] Heuristic search algorithms, on the other hand, solve random instances of problems like the 24-puzzle (Korf & Taylor 1996) that contain 10^{25} different states.

This raises the question: is planning in the blocks world so much more difficult than solving N-puzzles? Planning problems are actually 'nearly decomposable' and hence should probably be simpler than puzzles of the same (state) complexity. Yet the numbers show exactly the opposite. The explanation that we draw is that decomposition alone, as used in divide-and-conquer strategies, is not a sufficiently powerful search device for planning. This seems confirmed by the recent planner of Kautz and Selman (1996) that using a different search method solves instances of blocks world problems with 19 blocks and 10^{19} states.

In this paper, we cast planning as a problem of heuristic search and solve random blocks world problems with up to 25 blocks and 10^{27} states (bw_large.e in Table 1). The search algorithm uses the heuristic function that is defined below.

An Heuristic for Planning Problems

The heuristic function $h_G(s)$ that we define below provides an estimate of the number of steps needed to go from a state s to a state s' that satisfies the goal G. A state s is a collection of ground atoms and an action a determines a mapping from any state s to a new state $s' = res(a, s)$. In STRIPS (Fikes & Nilsson 1971), each (ground) action a is represented by three sets of atoms: the add list $A(a)$, the delete list $D(a)$ and the precondition list $P(a)$, and $res(a, s)$ is defined as $s - D(a) + A(a)$ if $P(a) \subseteq s$. The heuristic does not depend on the STRIPS representation and, indeed, later on we move to a different representation scheme. Yet in any case, we assume that we can determine in a straightforward way whether an action a makes a certain (ground) atom p true provided that a collection C of atoms are true. If so, we write $C \to p$. If actions are represented as in STRIPS, this means that we will write $C \to p$ when for an action a, p belongs to $A(a)$ and $C = P(a)$.

Assuming a set of 'rules' $C \to p$ resulting from the actions to be considered, we say that an atom p is *reachable* from a state s if $p \in s$ or there is a rule $C \to p$ such that each atom q in C is reachable from s.

The function $g(p, s)$ defined below, inductively assigns each atom p a number i that provides an estimate of the steps needed to 'reach' p from s. That is, $g(p, s)$ is set to 0 for all atoms p that are in s, while $g(p, s)$ is set to $i + 1$, for $i > 0$, for each atom p for which a rule $C \to p$ exists such that $\sum_{r \in C} g(r, s) = i$:

$$g(p,s) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } p \in s \\ i+1 & \text{if for some } C \to p, \ \sum_{r \in C} g(r,s) = i \\ \infty & \text{if } p \text{ is not reachable from } s \end{cases}$$

For convenience we define the function g for *sets* of atoms C as:

$$g(C, s) \stackrel{\text{def}}{=} \sum_{q \in C} g(q, s)$$

IBM RS/6000 C10 with a 100 MHz PowerPC 601 processor. We thank Blum, Furst, Kautz and Selman for making the code of GRAPHPLAN and SATPLAN available. The code for ASP is available at http://www.eniac.com/~bbonet.

[3] Actually, the fourth entry for SATPLAN is an estimate from the numbers reported in (Kautz & Selman 1996) as the memory requirements for the SAT encoding of the last two problems exceeded the capacity of our machines.

[4] By search we mean search in the space of states as opposed to the search in the set of partial plans as done in non-linear planning (McAllester & Rosenblitt 1991).

[5] This has been our experience but we don't have a reference for this.

[6] See (Slaney & Thiébaux 1996) for an estimate of the sizes of block worlds planning search spaces.

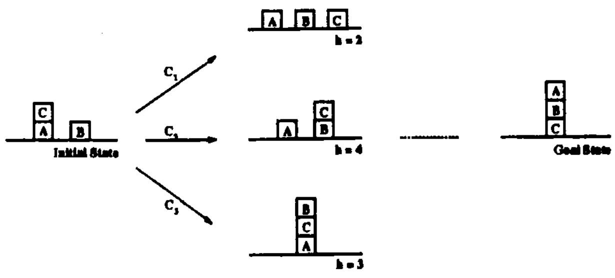

Figure 1: Heuristic for Sussman's Problem

and the heuristic function $h_G(s)$ as:

$$h_G(s) \stackrel{\text{def}}{=} g(G, s)$$

The heuristic function $h_G(s)$ provides an estimate of the number of steps needed to achieve the goal G from the state s. The reason that $h_G(s)$ provides only an *estimate* is that the above definition presumes that conjunctive goals are completely *independent;* namely that the cost of achieving them together is simply the sum of the costs of achieving them individually. This is actually the type of approximation that underlies decompositional planners. The added value of the heuristic function is that it not only decomposes a goal G into subgoals, but also provides estimates of the difficulties involved in solving them.

The complexity of computing $h_G(s)$ is linear in both the number of (ground) actions and the number of (ground) atoms. Below we abbreviate $h_G(s)$ as simply $h(s)$, and refer to $h(\cdot)$ as the *planning heuristic.*

Figure 1 illustrates the values of the planning heuristic for the problem known as Sussman's anomaly. It is clear that the heuristic function ranks the three possible actions in the right way pointing to (PUT-DOWN C A) as the best action. For example, to determine the heuristic value $h(s_3)$ of the state s_3 in which (ON B C) and (ON C A) hold relative to the goal in which (ON A B) and (ON B C) hold, we first determine the g-values of all atoms, e.g., $g((\text{ON B C}), s_3) = 0$, $g((\text{CLEAR B}), s_3) = 0$, ..., $g((\text{CLEAR C}), s_3) = 1$, ..., $g((\text{CLEAR A}), s_3) = 2$, ..., $g((\text{ON A B}), s_3) = 3$, ..., and hence $h(s_3)$ becomes the sum of $g((\text{ON B C}), s_3)$ and $g((\text{ON A B}), s_3)$, and thus $h(s_3) = 3$.

The Algorithms

The heuristic function defined above often overestimates the cost to the goal and hence is not *admissible* (Pearl 1983). Thus if we plug it into known search algorithms like A*, solutions will not be guaranteed to be optimal. Actually, A* has another problem: its memory requirements grows exponentially in the worst case. We thus tried the heuristic function with a simple N-best first algorithm in which at each iteration the first node is selected from a list ordered by increasing values of the function $f(n) = g(n) + h(n)$, where $g(n)$ is the number of steps involved in reaching n from the initial state, and $h(n)$ is the heuristic estimate associated with the state of n. The parameter N stands for the number of nodes that are saved in the list. N-best first thus takes constant space. We actually used the value N = 100.

problem	SATPLAN steps	SATPLAN time	N-best steps	N-best time
bw_large.a	6	0.7	8	1
bw_large.b	9	17.8	12	2
bw_large.c	14	524	21	40
bw_large.d	18	4,220	25	50

Table 2: Performance of N-best first compared with SATPLAN over some hard blocks world problems. Time is in seconds.

The results for some of the benchmark planning problems discussed in (Kautz & Selman 1996) are shown in Table 2, next to the the results obtained over the same problems using SATPLAN with direct encodings. The results show that the simple N-best first algorithm with a suitable heuristic function ranks as good as the most powerful planners even if the quality of the solution is not as good.

These results and similar ones we have obtained suggest than heuristic search provides a feasible and fruitful approach to planning. In all cases, we have found plans of reasonable quality in reasonable amounts of time (the algorithms are not optimal in either dimension). Yet, this amount of time that may be reasonable for off-line planning is not always reasonable for real time planning where an agent may be interacting with a dynamic world. Moreover, as we show below, such amounts of time devoted to making complete plans are often not needed. Indeed, we show below that *plans of similar quality can be found by agents that do not plan at all and spend less than a second to figure out what action to do next.*

To do that we turn to real time search algorithms and in particular to Korf's LRTA* (Korf 1990). Real time search algorithms, as used in 2-players games such as chess (Berliner & Ebeling 1989), interleave search and execution performing an action after a limited local search. They don't guarantee optimality but are fast and can react to changing conditions in a way that off-line search algorithms cannot.

LRTA*

A *trial* of Korf's LRTA* algorithm involves the following steps until the goal is reached:

1. *Expand:* Calculate $f(x') = k(x, x') + h(x')$ for each neighbor x' of the current state x, where $h(x')$ is the current estimate of the actual cost from x' to the goal, and $k(x, x')$ is the edge cost from x to x'. *Initially, the estimate $h(x')$ is the heuristic value for the state.*
2. *Update:* Update the estimate cost of the state x as follows:
$$h(x) \leftarrow \min_{x'} f(x')$$
3. *Move:* Move to neighbor x' that has the minimum $f(x')$ value, breaking ties arbitrarily.

The LRTA* algorithm can be used as a method for off-line search where it gets better after successive trials.

Indeed, if the initial heuristic values $h(x)$ are admissible, the updated values $h(x)$ after successive trials eventually converge to the true costs of reaching the goal from x (Korf 1990). The performance of LRTA* with the planning heuristic and the STRIPS action representation is shown in columns 5 and 6 of Table 3: LRTA* solves few of the hard problems and it then uses a considerable amount of time.

Some of the problems we found using LRTA* are the following:

- Instability of solution quality: LRTA* tends to explore unvisited states, and often moves along a far more expensive path to the goal than one obtained before (Ishida & Shimbo 1996).
- Many trials are needed to converge: After each move the heuristic value of a node is propagated to its neighbors only, so many trials are needed for the information to propagate far in the search graph.

A slight variation of LRTA*, that we call B-LRTA* (for bounded LRTA*), seems to avoid these problems by enforcing a higher degree of consistency among the heuristic values of nearby nodes before making any moves.

B-LRTA*

B-LRTA* is a true action selection mechanism, selecting good moves fast without requiring multiple trials. For that, B-LRTA* does *more* work than LRTA* before it moves. Basically it simulates n moves of LRTA*, repeats that simulation m times, and only then moves. The parameters that we have used are $n = 2$ and $m = 40$ and remain fixed for all the planning problems.

B-LRTA* repeats the following steps until the goal is reached:

1. *Deep Lookahead:* From the current state x, perform n simulated moves using LRTA*.
2. *Shallow Lookahead:* Still without moving from x, perform Step 1 m times always starting from state x.
3. *Move:* Execute the action that leads to the neighbor x' that has minimum $f(x')$ value, breaking ties randomly.

B-LRTA* is thus a recursive version of LRTA* that does a bit more exploration in the local space before each move, and usually converges in a much smaller number of trials. This local exploration, however, unlike the local min-min exploration in the standard version of LRTA* with lookahead (see (Korf 1990)) is not exhaustive. For that reason, we have found that B-LRTA* is able to exploit the information in the local search space more efficiently than LRTA* with lookahead. Indeed, in almost all the planning problems that we have considered (including different versions of the n-puzzle) and any lookahead depth, we have found that B-LRTA* achieves solutions with the same quality as LRTA* but in much smaller time (we hope to report these results in the full paper). Even more important for us, B-LRTA* seems to perform very well even after a *single* trial. Indeed, the improvement of B-LRTA* after repeated trials does not appear to be significant (we don't have an admissible heuristic).

We call the single trial B-LRTA* algorithm with the planning heuristic function, ASP for Action Selection for Planning. The performance of ASP based on the STRIPS representation for actions is displayed in columns 7 and 8 of Table 3. The time performance of ASP does not match the performance of SATPLAN, but what is surprising is that the resulting plans, computed in a single trial by purely local decisions, are close to optimal.

In the next section we show that both the time and quality of the plans can be significantly improved when the representation for actions is considered.

Representation

The representation for actions in ASP planning is important for two reasons: it affects memory requirements and the quality of the heuristic function.

Consider the STRIPS representation of an action schema like MOVE(x y z):

P: (ON x y) (CLEAR x) (CLEAR z)
A: (ON x z) (CLEAR y)
D: (ON x y) (CLEAR z)

standing for all the ground actions that can be obtained by replacing the variables x, y, and z by individual block names. In ASP planning this representation is problematic not only because it generates n^3 operators for worlds with n blocks, but mainly because it misleads the heuristic function by including ***spurious preconditions***. Indeed, the difficulty in achieving a goal like (ON x z) is a function of the difficulty in achieving the preconditions (CLEAR x) and (CLEAR z), but *not* the precondition (ON x y). The last atom appears as a precondition only to provide a 'handle' to establish (CLEAR y). But it does and should not add to the difficulty of achieving (ON x z).

The representation for actions below avoids this problem by replacing *relational* fluents by *functional* fluents. In the *functional representation,* actions are represented by a precondition list (P) as before but a new *effects* list (E) replaces the old add and delete lists. Lists and states both remain sets of atoms, yet all atoms are now of the form $t = t'$ where t and t' are terms. For example, a representation for the action (MOVE x y z) in the new format can be:

P: location(x) = y, clear(x) = true
clear(z) = true
E: location(x) = z, clear(y) = true
clear(z) = false

This new representation, however, does not give us much; the parameter y is still there, causing both a multiplication in the number of ground instances and the spurious precondition location(x) = y. Yet the functional representation gives us the flexibility to encode the action (MOVE x z) in a different way, using only two arguments x and z:

P: clear(x) = true clear(z) = true
E: location(x) = z, clear(z) = false,
clear(location(x)) = true

This action schema says that after moving x on top of z, the *new* location of x becomes z, the *new* location of

			direct		STRIPS encoding				functional encoding			
	GRAPHPLAN		SATPLAN		LRTA*		ASP		LRTA*		ASP	
problem	steps	time	steps	time	steps	time	steps	time	steps	time	steps	time
rocket_ext.a	34	268	34	0.17	—	—	52	82	28	6	28	6
rocket_ext.b	—	—	30	0.15	31	459	41	58	28	20	30	6
logistics.a	54	5,942	54	22	—	—	61	295	54		57	34
logistics.b	47	2,538	47	6	—	—	47	298	42		51	29
logistics.c	—	—	65	31	—	—	65	488	52		61	53
bw_large.a	6	4.6	6	0.7	8	60	8	33	8	2	8	1
bw_large.b	9	1,119	9	17.8	11	55	11	64	11	7	12	4
bw_large.c	—	—	14	524	—	—	—	—	19	31	18	14
bw_large.d	—	—	18	4,220	—	—	—	—	24	92	25	51
bw_large.e	—	—	—	—	—	—	—	—	35	1,750	36	417

Table 3: Performance of different planning algorithms. Time is in seconds. A blank space indicates that LRTA* didn't converge after 500 trials; best solution found is shown. A long dash (—) indicates that we were unable to complete the experiment due to memory limitations.

x is no longer clear, while the *old* location of x becomes clear.

We have used similar encodings for the other problems and the results of LRTA* and ASP over such encodings are shown in the last four columns of Table 3. Note that both algorithms do much better in both time and quality with functional encodings than with relational encodings. Indeed, both seem to scale better than SATPLAN over the hardest planning instances. The quality of the solutions, however, remain somewhat inferior to SATPLAN's. We address this problem below by adding an exploration component to the local search that precedes ASP moves.

The functional encodings are based on the model for representing actions discussed in (Geffner 1997), where both the language and the semantics are formally defined.

Execution

In this section we illustrate two features that makes ASP a convenient algorithm for real time planning: the possibility of working with a fixed time window, and the robustness in the presence of noise and perturbations.

Time for Action

There are situations that impose restrictions on the time available to take actions. This occurs frequently in real time applications where decision time is critical and there is no chance to compute optimal plans.

This kind of restriction is easy to implement in ASP as we just need to limit the time for 'deliberation' (i.e., lookahead search) before making a decision. When the time expires, the algorithm has to choose the best action and move.

Table 4 illustrates the results when such time limit is enforced. For each problem instance in the left column, the table lists the limit in deliberation time and the quality of the solutions found. Basically, in less than one second all problems are solved and the solutions found are very close to optimal (compare with Table 3 above). For times smaller than one second, the algorithm behaves as an anytime planning algorithm (Dean & Boddy 1988), delivering solutions whose quality gets better with time.

time limit	bw_large.a steps	bw_large.b steps	bw_large.c steps
0.05	18	46	—
0.10	9	18	119
0.25	9	12	81
0.50	9	12	20
1.00	9	12	18

Table 4: Quality of ASP plans as a function of a fixed time window for taking actions. Time is in seconds. A long dash (—) indicates that no solution was found after 500 steps.

Robustness

Most planning algorithms assume that actions are deterministic and are controlled by the planning agent. Stochastic actions and exogenous perturbations are usually not handled. ASP, being an action selection mechanism, turns out to be very robust in the presence of such perturbations.

Table 5 shows the results of running ASP in the bw_blocks.c problem using a very demanding type of perturbation: each time ASP selects an action, we force ASP to take a *different, arbitrary* action with probability p. In other words, when he intends to move, say, block A to block C, he will do another randomly chosen action instead, like putting B on the table or moving C to A, with probability p.

The results show how the quality of the resulting plans depend on the probability of perturbation p. It is remarkable that even when one action out of four misfires ($p = 0.25$), the algorithm finds solutions that are only twice longer that the best solutions in the absence of perturbations ($p = 0$). Actually, it appears that ASP may turn out to be a good planner in stochastic domains. That's something that we would like to explore in the future.

Learning and Optimality

We have also experimented with a simple strategy that makes the local exploration that precedes ASP moves less greedy. Basically, we added noise in the selection of the *simulated* moves (by means of a standard Boltzmann distribution and a temperature parameter

p	0.0	0.01	0.05	0.1	0.25	0.5	0.75
steps	18	18	19	24	39	64	—

Table 5: Quality of plans with perturbations with probability p (for bw_large.c). A long dash (—) indicates that no solution was found after 500 steps.

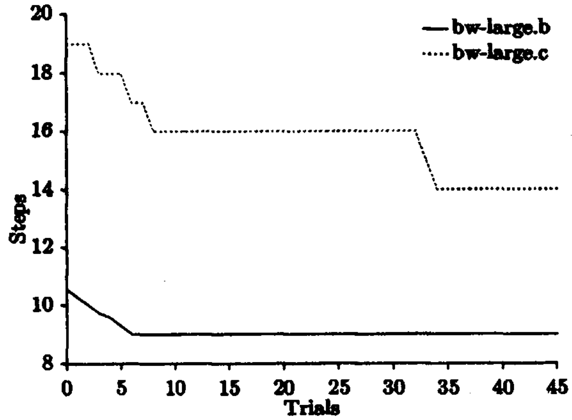

Figure 2: Quality of plans obtained after repeated trials of ASP with local randomized exploration.

that gradually cools off (Kaelbling, Littman, & Moore 1996)) and have found that while the quality performance of ASP in a single trial often decays slightly with the randomized local search (i.e., the number of steps to the goal), the quality performance of repeated trials of ASP tends to improve monotonically with the number of trials. Figure 2 shows this improvement for two instances of the blocks world, bw_large.b and bw_large.c, where optimal solutions to the goal are found after a few trials (7 and 35 trials respectively).

Summary

We have presented a real time algorithm ASP for planning that is based on a variation of Korf's LRTA* and a suitable heuristic function. ASP is robust and fast: it performs well in the presence of noise and perturbations and solves hard planning at speeds that compare well with the most powerful domain independent planners known to us. We also explored issues of representation and proposed an action representation scheme, different from STRIPS, that has a significant impact on the performance of ASP. We also experimented with randomized selection of the simulated moves and have found that the quality performance of ASP improves monotonically with the number of trials, until the optimal 'plans' are found.

A number of issues that we'd like to address in the future are refinements of the heuristic function and the representations. uses in off-line search algorithms and stochastic domains, and variations of the basic ASP algorithm for the solution of Markov Decision Processes (Puterman 1994). Indeed, the ASP algorithm (like Korf's LRTA*) turns out to be a special case of Barto's *et al.* Real Time Dynamic Programming algorithm (Barto, Bradtke, & Singh 1995), distinguished by an heuristic function derived from an action representation that is used for setting the initial state values

Acknowledgments

We thank Andreas Meier of the Laboratorio de Multimedia of the Universidad Simón Bolívar and Roger Bonet of Eniac, C.A. for the use of the machines in which the experiments were run. We also want to thank Bart Selman for many useful comments and pointers.

References

Agre, P., and Chapman, D. 1990. What are plans for? *Robotics and Autonomous Systems* 6:17–34.

Barto, A.; Bradtke, S.; and Singh, S. 1995. Learning to act using real-time dynamic programming. *Artificial Intelligence* 72:81–138.

Berliner, H., and Ebeling, C. 1989. Pattern knowledge and search: The suprem architecture. *Artificial Intelligence* 38:161–198.

Blum, A., and Furst, M. 1995. Fast planning through planning graph analysis. In *Proceedings of IJCAI-95.*

Brooks, R. 1987. A robust layered control system for a mobile robot. *IEEE J. of Robotics and Automation* 2:14–27.

Carbonell, J.; Blythe, J.; Etzione, O.; ; Gil, Y.; Joseph, R.; Kahn, D.; Knoblock, C.; and Minton, S. 1992. Prodigy 4.0: The manual and tutorial. Technical Report CMU-CS-92-150, CMU.

Chapman, D. 1987. Planning for conjunctive goals. *Artificial Intelligence* 32:333–377.

Dean, T., and Boddy, M. 1988. An analysis of time dependent planning. In *Proceedings AAAI-88*, 49–54.

Etzioni, O.; Hanks, S.; Draper, D.; Lesh, N.; and Williamson, M. 1992. An approach to planning with incomplete information. In *Proceedings of the Third Int. Conference on Principles of Knowledge Representation and Reasoning*, 115–125. Morgan Kaufmann.

Fikes, R., and Nilsson, N. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 1:27–120.

Geffner, H. and J. Wainer. 1997. A model for actions, knowledge and contingent plans. Technical report, Depto. de Computación, Universidad Simón Bolívar, Caracas, Venezuela.

Ishida, T., and Shimbo, M. 1996. Improving the learning efficiencies of realtime search. In *Proceedings of AAAI-96*, 305–310. Protland, Oregon: MIT Press.

Kaelbling, L.; Littman, M.; and Moore, A. 1996. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research* 4.

Kautz, H., and Selman, B. 1996. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proceedings of AAAI-96*, 1194–1201. Protland, Oregon: MIT Press.

Korf, R., and Taylor, L. 1996. Finding optimal solutions to the twenty-four puzzle. In *Proceedings of AAAI-96*, 1202–1207. Protland, Oregon: MIT Press.

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

Maes, P. 1990. Situated agents can have goals. *Robotics and Autonomous Systems* 6:49–70.

McAllester, D., and Rosenblitt, D. 1991. Systematic nonlinear planning. In *Proceedings of AAAI-91*, 634–639. Anaheim, CA: AAAI Press.

Newell, A., and Simon, H. 1972. *Human Problem Solving.* Englewood Cliffs, NJ: Prentice–Hall.

Nilsson, N. 1994. Teleo-reactive programs for agent control. *JAIR* 1:139–158.

Pearl, J. 1983. *Heuristics.* Morgan Kaufmann.

Penberthy, J., and Weld, D. 1992. Ucpop: A sound, complete, partiall order planner for adl. In *KR-92.*

Puterman, M. 1994. *Markov Decision Processes: Discrete Dynamic Stochastic Programming.* John Wiley.

Slaney, J., and Thiébaux, S. 1996. Linear time near-optimal planning in the blocks world. In *Proceedings of AAAI-96*, 1208–1214. Protland, Oregon: MIT Press.

Tyrrell, T. 1992. Defining the action selection problem. In *Proceedings of Simulation of Adaptive Behaviour.*

Veloso, M. 1992. *Learning by Analogical Reasoning in General Problem Solving.* Ph.D. Dissertation, Computer Science Department, CMU. Tech. Report CMU-CS-92-174.