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 (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.

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 LRITA* with this heuristic and introduce a variation of LRITA* 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

2 All algorithms are implemented in C and run on an
on the first problems but has trouble scaling up with the hardest block problems.\(^3\) \(AP\), 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, \(AP\) finds shorter sequential plans in the first two problems. On the other hand, the solutions found by \(AP\) in the last three problems are inferior to SATPLAN’s. In general \(AP\) does not guarantee optimal or close to optimal solutions, yet on the domain in which \(AP\) 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? 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 \(C \rightarrow p\). If actions are represented as in STRIPS, this means that we will write \(C \rightarrow p\) when for an action \(a\), \(p\) belongs to \(A(a)\) and \(C \subseteq P(a)\).

Assuming a set of ‘rules’ \(C \rightarrow p\) 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 \(C \rightarrow p\) 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 \(C \rightarrow p\) exists such that \(\sum_{r \in C} g(r, s) = i\):

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

For convenience we define the function \(g\) for sets of atoms \(C\) as:

\[
g(C, s) \overset{\text{def}}{=} \sum_{q \in C} g(q, s)
\]

\(^6\)See (Slaney & Thiébaux 1996) for an estimate of the sizes of block worlds planning search spaces.
junctive goals are completely
and the heuristic function $h_G(s)$ as:
$$h_G(s) \overset{\text{def}}{=} g(G, s)$$
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 reasoning that $h_G(s)$ provides only an esti-
mate is that the above definition presumes that con-
junctive 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 actu-
ally the type of approximation that underlies decom-
positional 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 sim-
ply $h(s)$, 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 deter-
mine 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 overesti-
mates 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 sim-
ple 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 associat-
ed with the state of $n$. The parameter $N$ stands for
the number of nodes that are saved in the list. N-best

The 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 1998).
- 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 \( \text{move}(x y z) \):

\[
P: \ (\text{on } x y ) \ (\text{clear } y ) \ (\text{clear } z) \\
A: \ (\text{on } x z ) \ (\text{clear } y) \\
D: \ (\text{on } x y ) \ (\text{clear } z)
\]

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 \((\text{on } x z)\) is a function of the difficulty in achieving the preconditions \((\text{clear } x)\) and \((\text{clear } z)\), but not the precondition \((\text{on } x y)\). The last atom appears as a precondition only to provide a ‘handle’ to establish \((\text{clear } y)\). But it does and should not add to the difficulty of achieving \((\text{on } x z)\).

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 \((\text{move } x y z)\) in the new format can be:

\[
P: \ \text{location}(x) = y, \ \text{clear}(z) = \text{true} \\
E: \ \text{location}(x) = z, \ \text{clear}(y) = \text{true} \ \\
\ \\
\text{clear}(z) = \text{false}
\]

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 \(\text{location}(z) = y\). Yet the functional representation gives us the flexibility to encode the action \((\text{move } x z)\) in a different way, using only two arguments \(x\) and \(z\):

\[
P: \ \text{clear}(z) = \text{true} \ \text{clear}(z) = \text{true} \\
E: \ \text{location}(x) = z, \ \text{clear}(z) = \text{false}, \ \text{clear(location}(x)) = \text{true}
\]

This action schema says that after moving \( z \) on top of \( z \), the new location of \( x \) becomes \( z \), the new location of
z is no longer clear, while the old location of z 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.

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

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.

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

<table>
<thead>
<tr>
<th>time limit</th>
<th>bw_large.a steps</th>
<th>bw_large.a steps</th>
<th>bw_large.a steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.05</td>
<td>18</td>
<td>46</td>
<td>---</td>
</tr>
<tr>
<td>0.10</td>
<td>9</td>
<td>18</td>
<td>119</td>
</tr>
<tr>
<td>0.25</td>
<td>9</td>
<td>12</td>
<td>81</td>
</tr>
<tr>
<td>0.50</td>
<td>9</td>
<td>12</td>
<td>20</td>
</tr>
<tr>
<td>1.00</td>
<td>9</td>
<td>12</td>
<td>18</td>
</tr>
</tbody>
</table>
Table 5: Quality of plans with perturbations with probability \( p \) (for \( \text{bw}_{\text{large.c}} \)). A long dash (—) indicates that no solution was found after 500 steps.

<table>
<thead>
<tr>
<th>( p )</th>
<th>0.0</th>
<th>0.01</th>
<th>0.05</th>
<th>0.1</th>
<th>0.25</th>
<th>0.5</th>
<th>0.75</th>
</tr>
</thead>
<tbody>
<tr>
<td>steps</td>
<td>18</td>
<td>18</td>
<td>19</td>
<td>24</td>
<td>39</td>
<td>64</td>
<td>—</td>
</tr>
</tbody>
</table>

Figure 2: Quality of plans obtained after repeated trials of \( \text{ASP} \) with local randomized exploration.

that gradually cools off (Kaelbling, Littman, & Moore 1996)) and have found that while the quality performance of \( \text{ASP} \) 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 \( \text{ASP} \) tends to improve monotonically with the number of trials. Figure 2 shows this improvement for two instances of the blocks world, \( \text{bw}\_\text{large.b} \) and \( \text{bw}\_\text{large.c} \), 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 \( \text{ASP} \) for planning that is based on a variation of Korf’s \( \text{LRTA}^* \) and a suitable heuristic function. \( \text{ASP} \) 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 \( \text{STRIPS} \), that has a significant impact on the performance of \( \text{ASP} \). We also experimented with randomized selection of the simulated moves and have found that the quality performance of \( \text{ASP} \) 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 \( \text{ASP} \) algorithm for the solution of Markov Decision Processes (Puterman 1994). Indeed, the \( \text{ASP} \) algorithm (like Korf’s \( \text{LRTA}^* \)) turns out to be a special case of Barto’s \textit{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


