A Robust and Fast Action Selection Mechanism for Planning*

Blai Bonet  Gábor Loerincs  Héctor Geffner

Departamento de Computación
Universidad Simón Bolívar
Apartado 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* (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 recently introduced by Kautz and Selman (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 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.1 SATPLAN performs very well

1All algorithms are implemented in C and run on an IBM RS/6000 C10 with a 100 MHz PowerPC 601 processor.
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.

<table>
<thead>
<tr>
<th>problem</th>
<th>steps</th>
<th>GRAPPI</th>
<th>SAT</th>
<th>ASP</th>
</tr>
</thead>
<tbody>
<tr>
<td>rocket_ext.a</td>
<td>34</td>
<td>268</td>
<td>0.1</td>
<td>28</td>
</tr>
<tr>
<td>logistics.b</td>
<td>47</td>
<td>2,538</td>
<td>6.0</td>
<td>51</td>
</tr>
<tr>
<td>bw_large.c</td>
<td>14</td>
<td>524</td>
<td>18</td>
<td>14</td>
</tr>
<tr>
<td>bw_large.d</td>
<td>18</td>
<td>4,220</td>
<td>25</td>
<td>51</td>
</tr>
<tr>
<td>bw_large.e</td>
<td></td>
<td></td>
<td></td>
<td>36</td>
</tr>
</tbody>
</table>

不同状态。5  故意搜索算法，在另一方面，解决随机实例中的问题，就像24-puzzle（Korf & Taylor 1996）一样包含10^{25}不同状态。

这提出了一个问题：在块世界中，与求解N-谜题相比，求解问题的世界更加困难吗？求解问题实际上是‘可分解的’，而在分而治之和结合策略中，没有一个充分强大的搜索设备用于求解。这似乎由最近由Kautz和Selman（1996）使用不同的搜索方法的块世界问题的实例所证实。为了19个块和10^{19}个状态。

在这个例子中，我们将求解问题看作是一个问题，求解随机的块世界问题，用最多25个块和10^{27}个状态（bw_large.e在表1中）。该搜索算法使用了已定义的 heuristic function。

### An Heuristic for Planning Problems

求解函数为h\_c(s)\(\text{def}\)我们下面为我们提供了一个例子，该估计的求解步数需要从一个状态s到一个状态s'来满足目标G。一个状态s是一个地上的原子和一个动作a确定了一个映射从任何状态s到一个状态s'，在 STRIPS（Fikes & Nilsson 1971），每个（ground）动作a被表示为三个原子的集合：添加列表A(a)，删除列表D(a)和预条件列表P(a)，以及res(a,s)被定义为s = D(a) + A(a)如果P(a) ∋ s。该 heuristic 不依赖于在 STRIPS 语言中的代表，事实上，后我们移动到一个不同的表示方案。

我们假设一个set of rules C → p 从行为的可考虑性来说，我们可以认为一个原子p是 reachable 从一个状态s如果p \in s或有一个规则 C → p 使得每个原子q在C是可从s reachable。如果动作是表示在STRIPS中，这意味着我们将写成 C → p 何时对于一个动作a，p属于A(a)和C = P(a)。

### 问答

这些问题并不总是精确地回答，但有几个数字是说明的。例如，领域独立的规划者，基于分而治之和结合策略的规划，当有较少的 heuristic functions 时，它会这样做：可分解的原子p true 提供了一个集合C的原子是 true。如果so，我们写C \rightarrow p。如果动作是表示在STRIPS中，这意味着我们将写成C → p 何时对于一个动作a，p属于A(a)和C = P(a)。

如果我们设置一个set of rules C → p 从行为的可考虑性来说，我们可以认为一个原子p是 reachable 从一个状态s如果p \in s或有一个规则 C → p 使得每个原子q在C是可从s reachable。如果动作是表示在STRIPS中，这意味着我们将写成C → p 何时对于一个动作a，p属于A(a)和C = P(a)。

### Planning as Search

规划问题是一个求解问题（Newell & Simon 1972）：有一个初始状态，存在操作，状态到后继状态的映射，而且存在目标状态必须要达到。然而求解是几乎从未以这种方式在任何教科书或研究中形成的。原因在于：待求解的问题的特定性质，它对于求解和分解以及没有良好的 heuristic functions。实际上，大部分工作都集中在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?

### 问答

这些问题并不总是精确地回答，但有几个数字是说明的。例如，领域独立的规划者，基于分而治之和结合策略的规划，当有较少的 heuristic functions 时，它会这样做：可分解的原子p true 提供了一个集合C的原子是 true。如果so，我们写C \rightarrow p。如果动作是表示在STRIPS中，这意味着我们将写成C → p 何时对于一个动作a，p属于A(a)和C = P(a)。

### 问答

这些问题并不总是精确地回答，但有几个数字是说明的。例如，领域独立的规划者，基于分而治之和结合策略的规划，当有较少的 heuristic functions 时，它会这样做：可分解的原子p true 提供了一个集合C的原子是 true。如果so，我们写C \rightarrow p。如果动作是表示在STRIPS中，这意味着我们将写成C → p 何时对于一个动作a，p属于A(a)和C = P(a)。

### An Heuristic for Planning Problems

求解函数为h\_c(s)\(\text{def}\)我们下面为我们提供了一个例子，该估计的求解步数需要从一个状态s到一个状态s'来满足目标G。一个状态s是一个地上的原子和一个动作a确定了一个映射从任何状态s到一个状态s'，在 STRIPS（Fikes & Nilsson 1971），每个（ground）动作a被表示为三个原子的集合：添加列表A(a)，删除列表D(a)和预条件列表P(a)，以及res(a,s)被定义为s = D(a) + A(a)如果P(a) ∋ s。该 heuristic 不依赖于在 STRIPS 语言中的代表，事实上，后我们移动到一个不同的表示方案。

我们假设一个set of rules C → p 从行为的可考虑性来说，我们可以认为一个原子p是 reachable 从一个状态s如果p \in s或有一个规则 C → p 使得每个原子q在C是可从s reachable。如果动作是表示在STRIPS中，这意味着我们将写成C → p 何时对于一个动作a，p属于A(a)和C = P(a)。

### 问答

这些问题并不总是精确地回答，但有几个数字是说明的。例如，领域独立的规划者，基于分而治之和结合策略的规划，当有较少的 heuristic functions 时，它会这样做：可分解的原子p true 提供了一个集合C的原子是 true。如果so，我们写C \rightarrow p。如果动作是表示在STRIPS中，这意味着我们将写成C → p 何时对于一个动作a，p属于A(a)和C = P(a)。

### An Heuristic for Planning Problems

求解函数为h\_c(s)\(\text{def}\)我们下面为我们提供了一个例子，该估计的求解步数需要从一个状态s到一个状态s'来满足目标G。一个状态s是一个地上的原子和一个动作a确定了一个映射从任何状态s到一个状态s'，在 STRIPS（Fikes & Nilsson 1971），每个（ground）动作a被表示为三个原子的集合：添加列表A(a)，删除列表D(a)和预条件列表P(a)，以及res(a,s)被定义为s = D(a) + A(a)如果P(a) ∋ s。该 heuristic 不依赖于在 STRIPS 语言中的代表，事实上，后我们移动到一个不同的表示方案。
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 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 putting \( c \) on \( A \), (PUTDOWN \( c \) \( A \)), as the best action.

### 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 \( N = 100 \).

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 that 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).

#### Table 2

<table>
<thead>
<tr>
<th>Problem</th>
<th>SATPLAN steps</th>
<th>SATPLAN time</th>
<th>N-best steps</th>
<th>N-best time</th>
</tr>
</thead>
<tbody>
<tr>
<td>bw_large.a</td>
<td>6</td>
<td>0.7</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>bw_large.b</td>
<td>9</td>
<td>17.8</td>
<td>12</td>
<td>2</td>
</tr>
<tr>
<td>bw_large.c</td>
<td>14</td>
<td>524</td>
<td>21</td>
<td>40</td>
</tr>
<tr>
<td>bw_large.d</td>
<td>18</td>
<td>4,220</td>
<td>25</td>
<td>50</td>
</tr>
</tbody>
</table>

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

\textbf{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 throughout the paper. 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.

The difference between B-LRTA\(^*\) and LRTA\(^*\) is that the former does a bit more exploration in the local space before each move, and thus usually converges in a much smaller number of trials. B-LRTA\(^*\) preserves some of the properties of LRTA\(^*\) as the convergence to optimal heuristic values after a sufficient number of trials when the initial heuristics are admissible. Yet more important for us, B-LRTA\(^*\) seems to perform very well 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 very 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.

\section*{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 \texttt{MOVE}(\(x\ y\ z\)):

\begin{verbatim}
P: (ON \(x\ y\)) (CLEAR \(x\)) (CLEAR \(z\))
A: (ON \(x\ z\)) (CLEAR \(y\))
D: (ON \(x\ y\)) (CLEAR \(z\))
\end{verbatim}

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 \textit{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 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 \textit{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 \texttt{(MOVE} \(x\ y\ z)\) in the new format can be:

\begin{verbatim}
P: location(x) = y, clear(x) = true
  clear(z) = true
E: location(x) = z, clear(y) = true
  clear(z) = false
\end{verbatim}

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 \((\text{MOVE} \(x\ z\))\) in a different way, using only two arguments \(x\) and \(z\):

\begin{verbatim}
P: clear(x) = true  clear(z) = true
E: location(x) = z, clear(y) = true
  clear(z) = false
  clear(location(x)) = true
\end{verbatim}

This action schema says that after moving \(x\) on top of \(z\), the new location of \(x\) becomes \(z\), the new location of \(x\) 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 LRRA\(^*\) 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), whereas both the language and the semantics are formally defined.

\section*{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.
Table 3: Performance of different planning algorithms. Time is in seconds. A blank space indicates that the planning algorithm 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 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.

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.

**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 limits are 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.

**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 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 Bolivar and Roger Donet of Eniac, C.A. for their support and computer time.

References