

International Journal on Artificial Intelligence Tools  
 © World Scientific Publishing Company

## STATE-SPACE PLANNING WITH VARIANTS OF A\*

AMOL DATATRAYA MALI

*Electrical Engineering & Computer Science, University of Wisconsin,  
 Milwaukee, WI 53211, United States of America  
 mali@uwm.edu*

MINH TANG

*Computer Science Department, Indiana University,  
 Bloomington, IN 47405-7104, United States of America  
 mhtang@cs.indiana.edu*

Received (22 May 2005)

Revised (14 February 2006)

Accepted (9 March 2006)

Significant advances have occurred in heuristic search for planning in the last eleven years. Many of these planners use A\*-style search. We report on five sound and complete domain-independent forward state-space STRIPS planners in this paper. The planners are AWA\* (Adjusted Weighted A\*), MAWA\* (Modified AWA\*), AWA\*-AC (AWA\* with action conflict-based adjustment), AWA\*-PD (AWA\* with deleted preconditions-based adjustment), and AWA\*-AC-LE (AWA\*-AC with lazy evaluation). AWA\* is the first planner to use node-dependent weighting in A\*. MAWA\*, AWA\*-AC, AWA\*-PD, and AWA\*-AC-LE use conditional two-phase heuristic evaluation. MAWA\* applies node-dependent weighting to a subset of the nodes in the fringe, after the two-phase evaluation. One novel idea in AWA\*-AC-LE is lazy heuristic evaluation which does not construct relaxed plans to compute heuristic values for all nodes. We report on an empirical comparison of AWA\*, MAWA\*, AWA\*-AC, AWA\*-PD, and AWA\*-AC-LE with classical planners AltAlt, FF, HSP-2 and STAN 4. Our variants of A\* outperform these planners on several problems. The empirical evaluation shows that heuristic search planning is significantly benefitted by node-dependent weighting, conditional two-phase heuristic evaluation and lazy evaluation. We report on the insights about inferior performance of our planners in some domains using the notion of *waiting time*. We discuss many other variants of A\*, state-space planners and directions for future work.

*Keywords:* Planning, A\*, Search

### 1. Introduction

Significant advances have occurred in heuristic search for planning in the last eleven years. These are clear from the success of planners FF<sup>1</sup>, HSP-2<sup>2</sup>, and AltAlt<sup>3</sup> at the international planning competitions in 2000<sup>4</sup>, 2002 and 2004. HSP-2 and AltAlt carry out weighted A\* (WA\*) style search. WA\*-style search has also been used by planner Sapa for planning in domains containing metric time and resource quantities.<sup>5</sup> FF carries out local search or weighted A\* (WA\*) search, depending

on the version chosen. FF, HSP-2, Sapa and AltAlt keep the weight in WA\* fixed throughout the search. The notion of changing the weight of the term contributing to the estimated path costs exists in the previous literature<sup>6,7,8</sup>, however it has not been applied to state-space planning. All of our planners using different variants of A\* use node-dependent weights. Current planners use the same heuristic to evaluate all nodes in the fringe. If the heuristic values cannot be computed fast, solving time increases, with more nodes in the fringe. If the heuristic is not very informative, evaluating all nodes in the fringe with the heuristic is not useful. More informative heuristics are generally computationally demanding. So one can use more informative heuristics for a small number of nodes in the fringe and use a computationally cheap heuristic for many more nodes in the fringe. This idea is used in the conditional two-phase heuristic evaluation in MAWA\*, AWA\*-AC, AWA\*-PD and AWA\*-AC-LE. The notion of using different heuristics as a part of a multi-phase heuristic evaluation exists in previous work on heuristic search.<sup>9</sup> However this type of multi-phase heuristic evaluation has not been used in planning before.

AWA\* differs from other planners previously developed due to its use of dynamic weighting in A\* search. MAWA\* is a modification of AWA\*. MAWA\* uses conditional two-phase heuristic evaluation along with node-dependent weights. MAWA\* uses a computationally cheap heuristic in its first phase to evaluate all nodes in the fringe. In the second phase, it uses a more informative heuristic to re-evaluate a selected set of nodes in the fringe. After this, it uses node-dependent weights for a selected set of nodes in the fringe. AWA\*-AC and AWA\*-PD differ from each other due to different information used in the second phase of conditional two-phase heuristic evaluation. They both apply node-dependent weighting to all nodes unlike MAWA\*.

The plan synthesis time depends on the time for heuristic evaluation. The time spent on heuristic evaluation of given nodes goes down if some nodes are evaluated with a computationally very fast process. This can be done even when the heuristic evaluation is uni-phase. Hence this differs from conditional two-phase heuristic evaluation. Specifically, some nodes can be evaluated only with heuristic  $h_1$  and the remaining nodes can be evaluated only with heuristic  $h_2$  which can be used much faster than  $h_1$ . One approach is to use the information about already-evaluated siblings of a node  $n$  or a randomly-generated number, to decide whether to evaluate  $n$  with  $h_1$ . AWA\*-AC-LE planner uses this kind of evaluation. AWA\*-AC-LE is obtained by integrating lazy evaluation with AWA\*-AC. AWA\*-AC and AWA\*-PD find relaxed plans for all nodes in the fringe, to compute the heuristic value. AWA\*-AC-LE decides whether to compute relaxed plan for a node based on the heuristic's value for its siblings and a randomly generated number. AWA\*-AC-LE does not find relaxed plans for all nodes in the fringe, thus performing a lazy heuristic evaluation. All of our variants of A\* planners are capable of backtracking. The conditions for their completeness are same as the conditions for the completeness of A\* (finite

branching factor and positive action costs). Finite branching factor guarantees that the time needed to expand any node is finite. Positive action costs guarantee that the cost of a path will increase after actions are added to the path. This guarantees that paths with infinite actions are not generated. All variants use the cost of the path from the root node to the node  $n$  at the end of the path, in evaluating  $n$ . None of our variants does pure heuristic search.

We report on an empirical evaluation of our planners on eight domains. Two of these domains (TSP-2n and TSP-3) are new and they contain dead ends. Domains like blocks world and transportation logistics do not have any dead ends since every state leads to some other state. The variants of A\* outperform FF and HSP-2 on several problems from TSP-2n and TSP-3. Our results show that the node-dependent weighting, lazy evaluation and conditional two-phase heuristic evaluation have the potential to scale up heuristic search planning in challenging new domains.

This paper makes the following contributions:

- We report on five sound and complete planners AWA\*, MAWA\*, AWA\*-AC, AWA\*-PD and AWA\*-AC-LE. AWA\* is the first planner to use node-dependent weighting in A\* search. MAWA\* is the first planner to use conditional two-phase heuristic evaluation and node-dependent weighting for a selected set of nodes in the fringe. AWA\*-AC-LE is the first state-space planner to integrate lazy evaluation with global search.
- We report on an empirical evaluation of A\*, WA\*, AWA\*, and MAWA\* on several problems. The evaluation shows that both AWA\* and MAWA\* are significantly superior to A\* and WA\* and that MAWA\* is superior to AWA\*.
- We also report on an empirical comparison of AWA\*, MAWA\*, FF, AltAlt and STAN 4 on several problems from five domains. The evaluation shows that both MAWA\* and AWA\* significantly outperform AltAlt and STAN 4 on all problems from these domains. MAWA\* and AWA\* also outperform FF on several problems.
- Our empirical evaluation shows that AWA\*-AC outperforms HSP-2 and FF on several problems from TSP-2n and TSP-3 domains. AWA\*-AC-LE is orders of magnitude faster than AWA\*-AC on TSP-3.
- The variants of A\* use multiple parameters whose values are user-specified. We report on an extensive empirical evaluation of the five variants with many sets of parameter values. This shows that the performance of the variants does not change significantly despite a wide variation in the parameters' values. So a user does not need to have a good knowledge of the domain or the planner in order to obtain superior performance.
- We report on an empirical comparison of AWA\* with WA\* with its static weight  $w = 2,3,4,5,6$  and 7. This comparison shows that node-dependent weighting is superior to static weighting.

This paper is based on our previous work.<sup>10,11</sup> This paper integrates the work from these papers. This paper has many experimental results and several conclusions that these conference papers do not. This paper includes many new references and

their discussion. The paper is organized as follows. We provide a brief introduction to STRIPS planning in section 2. We also review heuristic search-based state-space planners that are algorithmically close to our planners in this section. We also explain the notions of two-phase heuristic evaluation and conditional two-phase heuristic evaluation in this section. We describe AWA\* and MAWA\* in sections 3 and 4 respectively. We describe the algorithms of AWA\*-AC and AWA\*-PD in section 5. We report on AWA\*-AC-LE in section 6. We report on an empirical evaluation of A\*, WA\*, AWA\*, MAWA\*, AWA\*-AC, AWA\*-PD and AWA\*-AC-LE and their comparison with FF, AltAlt, HSP-2 and STAN 4 in section 7. In this section we also report on an empirical comparison of AWA\* with  $w = 2,3,4,5,6$  and 7. We also report on the performance of multiple runs of the five variants of A\* with different values of their parameters in section 7. We discuss our work on more variants of A\*, related work and directions for future work in section 8. We report on the conclusions in section 9.

## 2. Background

In this section, we provide a brief introduction to STRIPS planning. We also describe planners AltAlt, FF and HSP-2. We explain some key notions like relaxed planning graph, relaxed plan and PAD-mutually-exclusive (mutex) actions.

### 2.1. STRIPS Planning

STRIPS planning is the problem of synthesizing an executable sequence of sets of actions to achieve goal state from a completely-specified initial state. The goal state may be partially-specified. Actions are ground instances of operators, e.g. action  $move(A, B, C)$  is a ground instance of operator  $move(x, y, z)$ . Operators are state-transforming functions. This action moves block  $A$  from the top of block  $B$  to the top of block  $C$ . There are  $O(n^3)$  actions derivable from  $move(x, y, z)$  when there are  $n$  blocks. An operator is specified by its name (e.g.  $move$ ), parameter list (e.g.  $(x, y, z)$ ), preconditions, add effects and delete effects. Preconditions and effects of actions are obtained by grounding the preconditions and effects of operators. Preconditions must be true before an action is executed. Add effects are propositions that are added (made true) by the action. Delete effects are the propositions that are deleted (made false) by the action. Preconditions of  $move(A, B, C)$  are  $clear(A)$ ,  $clear(C)$  and  $on(A, B)$ . Add effects of this action are  $clear(B)$ ,  $on(A, C)$  and the delete effects of this action are  $on(A, B)$ ,  $clear(C)$ .  $A(o_i)$  denotes the set of add effects of action  $o_i$ .  $D(o_i)$  denotes the set of delete effects of action  $o_i$ .  $P(o_i)$  denotes the set of preconditions of  $o_i$ . State of world  $Result(S, o_i)$  after  $o_i$  is applied in a completely-specified world state  $S$  ( $P(o_i) \subseteq S$ ) is  $(S - D(o_i)) \cup A(o_i)$ .

Preconditions and effects of actions in STRIPS language are function-free ground literals. World states in STRIPS planning are conjunctions of function-free ground literals. Quantifiers, negated preconditions, disjunctions, metric time, numeric variables, uncertainty and objective functions are not allowed in STRIPS. Despite these

restrictions, STRIPS planning has been widely studied. This is because the techniques for efficiently solving STRIPS problems can be adapted to solving planning problems in more-expressive languages. STRIPS planning problems have been solved in a variety of ways like heuristic search, hill climbing, satisfiability solving, integer-linear programming and graph search. Forward state-space planning (FSS), backward state-space planning (BSS) and partial-order planning are the oldest STRIPS planning algorithms. A *partial plan* is a set of constraints that may be refined into a plan, e.g. an action sequence that does not achieve goal is a partial plan which can be refined by the addition of actions to yield a plan. A parallel/non-linear plan allows concurrent actions at some/all steps. A linear plan allows only 1 action per step.

## 2.2. Search Terminology

**Optimal Solution:** Optimal solution is a solution with minimum cost. In many domains, the cost of a path is the sum of the costs of the actions along the path. In such domains, path costs are same as path lengths when all actions have equal cost. Our planners assume that path cost is same as path length, the number of actions along the path. An optimal solution is same as the cheapest solution.

**Admissible Heuristic:** A heuristic  $h$  is admissible if it does not overestimate the cost of cheapest path from any node to the goal. If  $h_1$  and  $h_2$  are admissible heuristics, then  $h_3(n) = \min(h_1(n), h_2(n))$  is also admissible. Then  $h_4(n) = \max(h_1(n), h_2(n))$  is also admissible. Heuristics that under-estimate the costs of cheapest paths too much can significantly slow down search.

**WA\*:** Weighted A\* is a variant of A\*.<sup>12</sup> The weighted variant of A\* uses the following path cost equation  $f(n) = g(n) + w * h(n)$ ,  $w > 0$ , where  $g(n)$  represents the cost of the path from the root node to node  $n$ , and the  $h(n)$  represents the estimate of the cost of the cheapest path from  $n$  to goal. The use of  $w > 1$  generally reduces the number of nodes expanded and this generally reduces the solving times as well. The loss of optimality with the use of  $w$  is bounded by a constant factor.

## 2.3. FF, HSP-2, STAN 4 & AltAlt

AltAlt<sup>13</sup>, HSP-2<sup>2</sup>, and FF<sup>1</sup> are some of the very efficient heuristic search planners developed recently. HSP-2 is a state-space planner which allows a user to choose forward or backward direction of search, the heuristic to be used and the non-negative weight  $w$  of  $h(n)$ .  $h(n)$  is the estimated cost of cheapest path to goal from node  $n$ . HSP-2 does weighted A\*-style search. HSP-2 ignores positive as well as negative interactions between subgoals, in its heuristic evaluation. STAN 4 is a planner based on Graphplan<sup>14</sup>. STAN 4 also uses symmetry exploitation, exploitation of mutex relations and a generic type analysis that is carried out by TIM. TIM is type inference module<sup>15</sup>. STAN 4 performed well at the 2000 planning competition. STAN 4 and our variants have significant algorithmic differences.

**FF:** The development of FF was inspired by HSP, the initial version of HSP-2. FF is a forward state-space planner. Its initial version performs local search. A version doing weighted A\* search was developed later. FF considers positive interactions between subgoals while computing  $h(n)$ .  $h(n)$  in FF is the number of actions in the relaxed plan at node  $n$ . The relaxed plan at node  $n$  is found by backward search on the relaxed planning graph at node  $n$ . The notion of planning graph was introduced by Graphplan<sup>14</sup>. A relaxed planning graph (RPG) is constructed assuming that the delete effect lists of actions are empty. A RPG has alternating action levels and proposition levels like the planning graph<sup>14</sup>. A relaxed planning graph is shown in Fig. 1. There are several differences between the planning graphs used by FF and the planning graph in Graphplan. They are as follows: (i) Graphplan constructs only 1 planning graph and iteratively searches it and extends it, in order to find a plan. FF constructs many planning graphs, one for each node in its search tree. (ii) Graphplan takes into account the delete effects of actions and negative interactions among actions, when constructing and searching the planning graph. FF ignores delete effects when constructing and searching its RPGs. (iii) A RPG is constructed and searched much faster than Graphplan's planning graph. (iv) FF constructs RPG to find heuristic value. Graphplan constructs planning graph to find a plan. A detailed explanation of RPG and relaxed plans is given in section 2.4.

The initial version of FF (same as its 2000 competition version) does enforced hill climbing. This means that whenever  $h()$  values of all children of  $n$  are greater than or equal to  $h(n)$ , FF does breadth-first search, until a descendent of  $n$  with a better evaluation than  $h(n)$  is found. When such a descendent  $n'$  is found, FF includes the path from  $n$  to  $n'$  in its partial plan, removing all other nodes generated by breadth-first search from its memory. Enforced hill climbing differs from hill climbing in several ways. Hill climbing does not look beyond the children of the current state. Enforced hill climbing looks much deeper when it does breadth-first search. Hill climbing terminates when it reaches a peak where no neighbor has a better value. So it can get stuck in a local maximum easily. Enforced hill climbing is guaranteed to find a descendent of  $n$  with better evaluation than  $n$ , if such a descendent exists, due to the breadth-first search. Such a guarantee is not provided by hill climbing and its variations like stochastic hill climbing, first-choice hill climbing and random-restart hill climbing. Random-restart hill climbing conducts a series of hill-climbing searches from randomly generated initial states. Such multiple searches from different initial states are not used by enforced hill climbing. Overall, FF performed significantly better than HSP-2 and AltAlt at the 2000 international planning competition. AltAlt uses much more informative heuristics than HSP-2 and FF. Many of AltAlt's heuristics take into account positive as well as negative interactions between subgoals. The heuristics are derived from a planning graph. AltAlt as well as FF did very well at the 2000 competition. FF version 1.0 was awarded the Exceptional Performance Award for domain independent planners in this competition.<sup>4</sup>

#### 2.4. Mutual Exclusion & Relaxation in Planning

MAWA\* and AWA\*-AC use the notion of PAD-mutex actions. These are binary action-action mutexes found by a comparison of preconditions, add effects and delete effects (hence the name PAD-mutex). The utility of such mutex actions in planning was first demonstrated by Graphplan.<sup>14</sup> Two actions  $o_i$  and  $o_j$  are PAD-mutex if (i)  $o_i$  deletes some precondition of  $o_j$  or (ii)  $o_j$  deletes some precondition of  $o_i$  or (iii)  $o_i$  needs  $p$  and  $o_j$  needs  $\neg p$  or (iv)  $o_i$  adds  $p$  and  $o_j$  deletes  $p$  or (v)  $o_i$  deletes  $p$  and  $o_j$  adds  $p$ .

**Relaxed planning graph (RPG):** The notion of RPG was introduced in the work on planner FF.<sup>1</sup> An RPG is computed as an aid to compute the heuristic values in FF. We denote the goal of a planning problem by  $G$  in the rest of the paper. A subgoal from  $G$  is a predicate or proposition from  $G$ . A RPG is constructed assuming that the delete effect lists of actions are empty. A RPG has alternating action levels and proposition levels like the planning graph.<sup>14</sup> The construction of a RPG stops when a proposition level containing all subgoals from  $G$  is found or when the proposition level does not change, whichever occurs earlier. RPGs are constructed in time polynomial in the number of actions and propositions. First proposition level of RPG of node  $n$  is same as the world state in  $n$ . First action level of RPG of node  $n$  is same as the set of actions whose preconditions appear in the first proposition level. In general  $i$  th action level in RPG where  $i > 1$  is the set of actions whose preconditions appear in  $i$  th proposition level and which do not occur in  $i - 1$  th action level.  $i$  th proposition level is the union of the  $i - 1$  th proposition level and the effects of the actions in  $i - 1$  th action level. If  $i$  th proposition level is same as  $i + 1$  th proposition level, and the  $i$  th proposition level does not contain all subgoals from  $G$ , the construction of relaxed planning graph is terminated with the conclusion that  $G$  is not achievable from the node. A relaxed planning graph is shown in in Fig. 1. The initial state for this problem is  $(a \wedge b \wedge e)$  and the goal is  $(m \wedge e \wedge d)$ . The 11 action descriptions are in the form (preconditions)(action)(effects), shown in the left half of the figure. This relaxed planning graph has 3 proposition levels and 2 action levels.

**Relaxed plan:** This notion was introduced by FF. It is found by backward search on a relaxed planning graph.<sup>1</sup> A relaxed plan at node  $n$  is a subgraph of the RPG at node  $n$ . A relaxed plan at  $n$  achieves goal from  $n$  in absence of delete effects. A relaxed plan is found in polynomial time. Actions in the  $i$  th step of a relaxed plan for node  $n$  are a subset of the actions in the  $i$  th action level of the RPG for  $n$ . Consider a RPG with  $k + 1$  proposition levels and  $k$  action levels. The backward search for finding relaxed plan selects an action from action level  $k$  to achieve each subgoal from  $G$  in proposition level  $k + 1$  that does not appear in proposition level  $k$ . These actions are the actions that occur in the last step of the relaxed plan at node  $n$ . The set of subgoals in proposition level  $k$  is then the union of the following two sets - (i) the union of the preconditions of actions in the  $k$  th step of relaxed plan, and (ii) the subgoals in proposition level  $k + 1$  that also appear in proposition

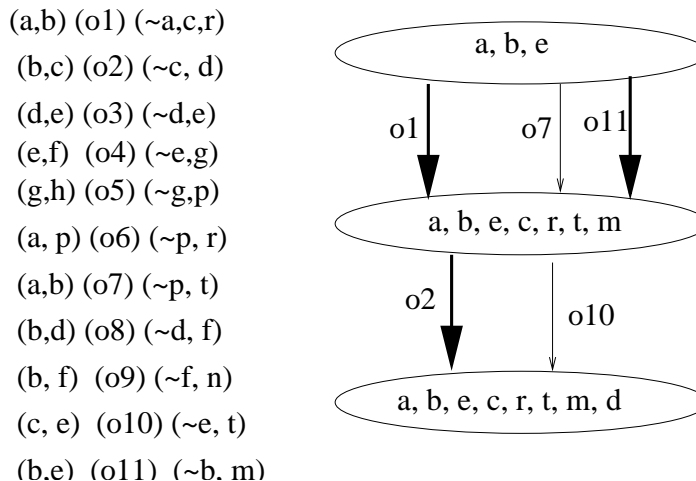


Fig. 1. Relaxed planning graph and relaxed plan

level  $k$ . Actions in the  $i$ th step of relaxed plan achieve only those subgoals in proposition level  $i + 1$  that do not appear in proposition level  $i$ . The backward search terminates when no subgoals remain to be achieved. Note that the number of steps in the relaxed plan is a lower bound on the number of steps in the plan that achieves the goal from node  $n$ . The actions in the relaxed plan for the problem in Fig. 1 are shown with thick arrow heads. The relaxed plan is  $\{\{o1, o11\}, \{o2\}\}$ . A plan for this problem is  $[o1, o2, o11]$ .

**Two-phase heuristic evaluation:** In this scheme, nodes are heuristically evaluated in two phases. In the first phase, heuristic  $h_1$  is used to evaluate the nodes. In the second phase, heuristic  $h_2$  is used to evaluate the nodes. The  $h$  value of a node depends on the values of  $h_1$  and  $h_2$ . For example,  $h(n)$  may be  $h_1(n) + h_2(n)$  or it may be  $\max(h_1(n), h_2(n))$ . The phased relaxation in Sapa<sup>5</sup> can be viewed as two-phase heuristic evaluation. In the first phase Sapa does not consider numeric variables. In the second phase, it does consider numeric variables. Two-phase heuristic evaluation differs from the combination of heuristics in the pattern database approach.<sup>16,17</sup>  $h_1$  and  $h_2$  for various world states are precomputed and the values are stored in tables called pattern databases.<sup>16,17</sup> In the pattern database approach, there is no online evaluation of nodes in two phases.  $h_1$  and  $h_2$  can be computed in any order or in parallel, in the pattern database approach.<sup>16,17</sup> Pattern database approach allows a combination of more than two heuristics.

**Conditional two-phase heuristic evaluation:** This differs from two-phase evaluation. The two-phase evaluation applies two phases to all nodes in the fringe. The conditional two-phase heuristic evaluation re-evaluates only a small subset of the nodes in the fringe in the second phase. The nodes to be evaluated in the second phase are selected based on the evaluation from phase 1. Hence the evaluation in second phase is “conditional”. The second phase may detect less promising nodes

that phase 1 did not. This idea can be extended to conditional multi-phase heuristic evaluation.<sup>18</sup>

**Considering information ignored in the computation of relaxed plans:** The information ignored in the construction of relaxed plans can be used in the future phases of a multi-phase heuristic evaluation. MAWA\*, AWA\*-AC, AWA\*-PD and AWA\*-AC-LE use some information ignored in the phase 1 evaluation in phase 2. These four planners ignore delete effects of actions in the synthesis of relaxed plans in phase 1 and consider the delete effects in phase 2. This helps in an efficient plan synthesis, as our results show.

### 3. AWA\*

It's a well known fact<sup>9</sup>, that the A\* algorithm<sup>19,12</sup> might spend too much time choosing over states with approximately equal costs. This leads to a high search time in many cases. The weighted variant of A\* attempts to remedy this situation by adjusting the estimated distances to goal. The adjustment is carried out by multiplying the estimated cost of cheapest path to goal from a node by a positive number (weight). This generally reduces the number of nodes expanded and this generally reduces the solving time as well. Adjusted weighted variant of A\* (AWA\*) attempts to compensate for the inaccuracy of the heuristic value, as well as improve the optimality of WA\*, while trying to make sure that the search time is still acceptable.

The A\* search algorithm<sup>19</sup> uses the following cost equation

$$f(n) = g(n) + h(n)$$

where  $g(n)$  represents the cost of the path from the root node to node  $n$ , and the  $h(n)$  represents the estimate of the cost of the cheapest path from  $n$  to the goal. A\* always expands node with lowest value of  $f$  first. Our implementations of A\* and its variants construct search trees. A\* is complete if the branching factor is finite and all operator costs are positive. If  $h$  is admissible, then A\* is optimal.

#### A\* algorithm

1. Expand the root node  $r$ . Compute  $g(n)$ ,  $h(n)$  and  $f(n)$  for each child  $n$  of the root. Put all children of the root into fringe.
- 2(a). Apply goal test to the node  $n$  from the fringe with lowest  $f$  value. If goal is true in  $n$ , return the path from  $r$  to  $n$ , else go to 2(b).
- 2(b). Expand  $n$ . Compute  $g(n')$ ,  $h(n')$  and  $f(n')$  for each child  $n'$  of  $n$ . Add all children of  $n$  to the fringe. Go to 2(a).

AWA\* uses the following equation

$$f(n) = \varphi(n) * g(n) + \psi(n) * h(n)$$

where  $\varphi(n)$  and  $\psi(n)$  are functions that depend on the current state (world state in node  $n$ ). So the weights used in AWA\* are dynamic, and state-dependent. Dynamic

weighting for  $g(n)$  and  $h(n)$  in A\* was first proposed in 1973<sup>6</sup>, but it was never used in AI planning. Pohl proposed a version of A\* using  $f(n) = w_g * g(n) + w_h * h(n)$ , where  $w_g$  and  $w_h$  are adjusted as the search progresses. The weighting in AWA\* is node-dependent. The weighting<sup>6</sup> is not node-dependent.

A version of A\* using dynamic weighting for  $h()$  is mentioned on page 88.<sup>9</sup> This version uses the following equation

$$f(n) = g(n) + h(n) + \epsilon \left[ 1 - \frac{d(n)}{N} \right] h(n)$$

where  $d(n)$  is the depth of node  $n$  and  $N$  is the depth of the shallowest state where the goal is true. A limitation of this approach is that we must know  $N$ , or at least be able to estimate it with a high degree of accuracy. In Pearl's scheme, all nodes at same depth always have the same weight for their  $h()$ , which is not the case in AWA\*.

In the implementation of AWA\*, MAWA\*, AWA\*-AC, AWA\*-PD, and AWA\*-AC-LE, we chose the functions  $\varphi(n)$  and  $\psi(n)$  as follows  $\varphi(n) = \mu_g$ ,  $\psi(n) = \mu_h - \frac{s(n)}{\delta_h * |G|}$ , where  $\mu_g, \mu_h$ , and  $\delta_h$  are positive constants,  $s(n)$  is the number of subgoals true in the state in node  $n$ , and  $|G|$  is the number of subgoals in the goal of the planning problem. Each subgoal is a ground, function-free predicate. We also require  $\mu_h > \frac{1}{\delta_h}$  so that  $\psi(n)$  is always positive. The above choice of  $\phi(n)$  and  $\psi(n)$  has the following properties: (i) the value of  $\psi(n)$  is computed fast, and (ii)  $\psi(n)$  adjusts the  $h()$  value. With  $\psi(n)$ , nodes with more achieved subgoals are likely to be expanded earlier, when  $g()$  and  $h()$  values of the nodes are approximately equal. This arises from the fact that as  $s(n)$  increases,  $\frac{s(n)}{\delta_h * |G|}$  also increases and  $\psi(n)$  decreases which decreases  $f(n)$ .

#### 4. MAWA\*

MAWA\* uses a variation of dynamic weighting which uses weights for only more-promising nodes. The set of these more-promising nodes is the *candidate set*. A factor that can adversely affect the performance of AWA\* is the use of same heuristic to compute  $h()$  values of all nodes. If a heuristic is not very informative or is time-consuming to compute, it may not be worth evaluating all nodes with it. One way to control the time for heuristic evaluation and also get more information about the nodes is to use a conditional two-phase heuristic evaluation. In this, all nodes in fringe are evaluated by some heuristic in phase 1 and some of these are evaluated by some other heuristic in phase 2. MAWA\* uses both conditional two-phase heuristic evaluation and node-dependent weights for only selected nodes.

In our MAWA\* implementation,  $h(n)$  is same as the number of actions in the relaxed plan at  $n$ , as in FF.<sup>1</sup>  $h'(n)$  is the heuristic value of  $n$  if second phase of conditional two-phase heuristic evaluation is applied to  $n$ . In the MAWA\* implementation,  $h'(n) = h(n) + \sum_{i=1}^{k(n)} m_i(n)$ , where  $k(n)$  is the number of steps in the relaxed plan at node  $n$  and  $m_i(n)$  is the number of pairs of PAD-mutex actions in

the  $i$  th step of the relaxed plan at  $n$ . We refer to  $\sum_{i=1}^{k(n)} m_i(n)$  as  $PAD - sum(n)$ . This adjustment gives us some more information. A node with a high number of PAD mutexes in its relaxed plan might not be a good node, even though its  $h()$  value might be lower than that of other nodes. The adjustment to  $h(n)$  is based on the intuition that more actions than the ones in relaxed plan at  $n$  will be generally needed to achieve goal from  $n$ , whenever the relaxed plan has PAD-mutex actions at same steps. These additional actions are needed to re-establish preconditions or effects that are deleted by mutex actions.

### MAWA\* algorithm

- 1(a). **(Phase 1)** For every node  $n$  in the fringe that was not evaluated before, compute  $f(n) = g(n) + h(n)$ .
- 1(b). **(Phase 2)** If  $h(n) < c$ , where  $c$  is a pre-specified constant, adjust  $h(n)$  to  $h'(n)$  where  $h'(n) = h(n) + PAD - sum(n)$  and compute  $f(n) = g(n) + h'(n)$ .
2. Find the node  $n_{min}$  in the fringe with minimal  $f()$  value.
3. Let the candidate set  $S$  be the set of nodes  $n$  in the fringe such that  $f(n)$  is less than or equal to  $(1 + \epsilon) * f(n_{min})$ , where  $\epsilon$  is a small positive constant.
4. For every  $n \in S$ , compute  $(\mu_g * g(n) + (\mu_h - \frac{s(n)}{(\delta_h * |G|)}) * h(n))$
5. Expand node  $n_x \in S$  for which  $(\mu_g * g(n_x) + (\mu_h - \frac{s(n_x)}{(\delta_h * |G|)}) * h(n_x))$  value is minimal among all the nodes from  $S$ .
6. Go to step 1(a) if no plan is found, else return the plan.

MAWA\* uses five user-specified parameters which are  $\mu_g, \mu_h, \delta_h, \epsilon$  and  $c$ . MAWA\* is inspired by the  $A_e^*$  scheme.<sup>9</sup> Pearl<sup>9</sup> mentions that one can use a heuristic  $h_1$  to form candidate set  $S$  and a heuristic  $h_2$  to select a member of  $S$  to expand next. MAWA\* can be viewed as using  $h_1$  and  $h_2$ .  $h_1$  in MAWA\* uses  $\epsilon$  and  $f_{min}$  values to form  $S$ .  $h_2$  in MAWA\* uses node-dependent weights.

## 5. AWA\*-AC and AWA\*-PD

In this section, we describe the algorithms of planners AWA\*-AC and AWA\*-PD. The algorithm of AWA\*-AC has six steps. First two steps are about conditional two-phase heuristic evaluation. Step 2 which is about second phase, carries out an adjustment to  $h()$  values found in first phase, if  $h(n) < c$ , where  $c$  is a user-specified positive constant. For example, if relaxed plan of a node  $n$  with  $h(n) = 7$  (from phase 1) has 4 pairs of PAD-mutex actions, one in its step 1 and 3 in step 3, and  $c = 10$ , then the  $h(n)$  after the second phase will be 11 ( $= 7 + 1 + 3$ ). This adjustment can give us some more information. Note that step 2 considers some information (delete effects and PAD-mutexes) that is ignored in the computation of relaxed plan. AWA\*-AC differs from AWA\* due to its use of conditional two-phase heuristic evaluation. AWA\*-AC differs from MAWA\* due to the fact that AWA\*-AC does not use candidate set.

**AWA\*-AC algorithm**

1. Evaluate all nodes in the fringe whose  $h$  value was not found, by computing  $h(n)$ , where  $h(n)$  = number of actions in the relaxed plan at  $n$ .
2. For all nodes in the fringe { If  $h(n) < c$ , and  $n$  was not evaluated in second phase before, then  $h(n) = h(n) + PAD - sum(n)$  }.
3. For all nodes in the fringe whose  $f$  value has not been found, compute  $f(n) = \mu_g * g(n) + (\mu_h - \frac{s(n)}{\delta_h * |G|}) * h(n)$
4. Expand node in the fringe with lowest  $f()$  value.
5. Go to 1 if no plan is found.
6. Return plan.

AWA\*-PD uses a different adjustment strategy than AWA\*-AC in phase 2 of the conditional two-phase heuristic evaluation. Its algorithm too has 6 steps. AWA\*-PD differs from AWA\*-AC only in second step. Though the delete effects of actions are ignored in first phase, they are considered in the second phase for the purpose of heuristic adjustment. Several actions in a relaxed plan for a node generally appear in the plan for the node. So negative interactions between actions in the relaxed plan are relevant to generating a plan from the same node. New actions may be needed to establish the deleted preconditions of actions in a relaxed plan, if these actions also occur in plan. In a worse case, one new action is needed to establish each deleted precondition. This is the motivation for the additive term in the heuristic adjustment in step 2 of AWA\*-PD. AWA\*-AC and AWA\*-PD both use four user-specified parameters which are  $\mu_g, \mu_h, \delta_h$  and  $c$ .

**AWA\*-PD algorithm**

2. For all nodes in the fringe { If  $h(n) < c$ , and  $n$  was not evaluated in second phase, then  $h(n) = h(n) +$  Total number of preconditions of actions in the relaxed plan at  $n$  that are deleted by some action in the relaxed plan }.

**6. AWA\*-AC-LE**

In this section we describe the algorithm used by AWA\*-AC-LE. AWA\*-AC-LE decides whether to compute relaxed plan for a node based on the number of its interesting and less interesting siblings and a randomly generated number. AWA\*-AC-LE is motivated by the intuition that constructing relaxed plans for fewer nodes may speed up plan synthesis significantly.

We explain the notation used in the description of the algorithm next.  $parent(n)$  is parent of node  $n$ .  $\theta_1$  is the limit for number of interesting siblings.  $\theta_2$  is the limit for the number of less interesting siblings.  $n_i$  is  $i$  th child of node  $n$ . A node  $n$  is interesting if  $h(n) < h(parent(n))$ . A node  $n$  is less interesting if  $h(n) > h(parent(n))$ .  $IS(n)$  is the number of interesting siblings of node  $n$ .  $LIS(n)$  is the number of less

interesting siblings of node  $n$ .  $p$  is a user-specified positive integer used in deciding whether to compute relaxed plan for a node.  $\theta_1$  and  $\theta_2$  are user-specified. Steps 1,2,3 and 4 of the algorithm are about heuristic evaluation of nodes. Step 5 is about the second phase of conditional two-phase heuristic evaluation. Step 6 is about computing  $f()$  values using node-dependent weights.

### AWA\*-AC-LE algorithm

1. Expand the root node. Find the  $h$  value for root node using the relaxed plan heuristic.
2. Let  $n$  be the most recently expanded node. Let  $nc$  be the number of children of  $n$ . If  $nc \leq \min(\theta_1, \theta_2)$ , then evaluate all children using relaxed plan heuristic.
3. If  $nc > \min(\theta_1, \theta_2)$ , then evaluate first  $\min(\theta_1, \theta_2)$  children using the relaxed plan heuristic. (Note:  $i$  th child of  $n$  is the  $i$  th node generated while expansion of  $n$ .)
4. For  $i = (\min(\theta_1, \theta_2) + 1)$  to  $nc$ 
  - {
  - If  $((IS(n_i) < \theta_1) \wedge (LIS(n_i) < \theta_2))$
  - then  $h(n_i) =$  Number of actions in the relaxed plan at  $n_i$
  - Else
  - { Generate a random positive integer  $x$ .
  - If  $((x \text{ modulo } p) == 0)$ , then
  - $h(n_i) =$  Number of actions in the relaxed plan at  $n_i$
  - Else
  - $h(n_i) = h(n) + \alpha$
  - }
  - }
5. For  $(i = 1$  to  $nc)$  { If  $((n_i$  was evaluated using relaxed plan heuristic)  $\wedge$   $(h(n_i) < c) \wedge$   $(h(n_i)$  was not adjusted in second phase)), then  $h(n_i) = h(n_i) + PAD - sum(n)$  }
6. For all  $nc$  children of  $n$ , find  $f(n_i) = \mu_g * g(n_i) + (\mu_h - \frac{s(n_i)}{\delta_n * |G|}) * h(n_i)$
7. Expand node in the fringe with lowest  $f()$  value.
8. Go to 2 if no plan is found.
9. Return plan.

The motivation for step 4 is as follows. If a node has several interesting siblings ( $\geq \theta_1$ ), it too could be interesting. If a node has several less interesting siblings ( $\geq \theta_2$ ), it too could be less interesting. So if  $(IS(n) \geq \theta_1) \vee (LIS(n) \geq \theta_2)$ , then it may not be useful to compute relaxed plan for  $n$ . Hence a test based on a randomly-generated number is used in this case to decide whether to compute relaxed plan for  $n$ . If  $(IS(n) < \theta_1) \wedge (LIS(n) < \theta_2)$ , then there is a wider variation in the siblings of  $n$ . In this case it could be worth paying more computational cost to evaluate  $n$ .

Hence relaxed plan is found for  $n$  if  $(IS(n) < \theta_1) \wedge (LIS(n) < \theta_2)$ .  $\alpha$  denotes the user-specified positive constant used to compute the  $h$  value of a node fast. The behavior of step 4 is sensitive to the order in which nodes are generated, since the evaluation of a node is affected by the evaluation of its siblings generated before it. AWA\*-AC-LE uses eight user-specified parameters which are  $\mu_h, \mu_g, \delta_h, p, \alpha, \theta_1, c$  and  $\theta_2$ .

## 7. Empirical Evaluation

We have implemented forward state-space search planners using A\*, WA\*, AWA\*, MAWA\*, AWA\*-AC, AWA\*-PD and AWA\*-AC-LE in C++. All these planners terminate immediately after the first plan is found. The cost equation used by the implementation of AWA\* is

$$f(n) = 0.5 * g(n) + \left(2 - \frac{s(n)}{|G|}\right) * h(n)$$

Values of  $\delta_h, \mu_g$  and  $\mu_h$  were always respectively 1, 0.5 and 2 for all variants, unless stated otherwise. In A\*, WA\*, MAWA\*, AWA\*, AWA\*-AC-LE, AWA\*-AC and AWA\*-PD,  $h(n)$  is equal to the number of actions in the relaxed plan at node  $n$  unless it is adjusted by second phase or found by lazy evaluation. The empirical work is reported using 14 tables in this section. All the results from the first 12 tables are collected from a 1.8 GHz Athlon XP with 256MB RAM with Mandrake Linux 9. MAWA\* used  $\epsilon = 0.2$ . In A\*, WA\*, AWA\*, MAWA\*, AWA\*-AC, AWA\*-PD, and AWA\*-AC-LE,  $g(n)$  is same as the depth of node  $n$ . MAWA\* used  $c = 10$  for the evaluation in tables 4 and 5. AWA\*-AC, AWA\*-PD and AWA\*-AC-LE used  $c = 10$  unless stated otherwise.  $c$  is used to decide if  $h(n)$  should be adjusted in second phase. WA\* used  $w = 4$  in the equation  $f(n) = g(n) + w * h(n)$  for the evaluation in tables 1 and 2. Tables 1,2,3,4,..., 12 report the number of actions in the plans found. In the tables of results, “acts” denotes the number of actions in the plans found and “t” denotes the solving time. The solving times are reported in cpu seconds. An entry of - under the actions heading in tables 1,2,3,4,5 and 6 indicates that no plan was found within 10 CPU minutes. For STAN 4,<sup>15</sup> the results under the actions heading are of the form  $x/y$  where  $x$  is the number of time steps and  $y$  is the number of actions. STAN 4 is very good at finding parallel plans. All other planners used in the empirical evaluation find serial plans.

We used two different versions of FF in the empirical comparison. We used FF version 1.0 for domains that lack non-invertible actions. For domains that contain non-invertible actions, we used FF version 2.3 since FF version 1.0 would not handle these domains. TSP-2, TSP-2n and TSP-3 contain non-invertible actions. FF 1.0 terminated with the message that the domains had non-invertible actions. FF version 1.0 performs enforced hill climbing while searching for a solution. FF version 2.3 performs best first search after using enforced hill climbing, if needed. FF version 1.0 terminates without a solution once it has visited 150,000 states. FF version 2.3 does not have this limitation.

Planners AltAlt, FF, HSP-2 and STAN 4 were run using their default settings. Default settings of HSP-2 are forward direction,  $w = 2$  and sum heuristic. The default settings of the planners are generally quite robust and give good performance. Both versions of FF are implemented in C. STAN 4 is implemented in C++ and the version of AltAlt that we used is also implemented in C++.

### 7.1. Domain Descriptions

We used several domains in the evaluation. The domains are explained below.

- **Blocks World:** This is blocks domain using the  $move(x,y,z)$ ,  $move(x,Table,y)$  and  $move(x,y,Table)$  operators.  $move(x,y,z)$  moves block  $x$  from the top of block  $y$  to the top of block  $z$ .  $move(x,Table,y)$  moves block  $x$  from the table to the top of block  $y$ .  $move(x,y,Table)$  operator moves block  $x$  from the top of block  $y$  to the table.
- **Blocks inversion:** This is a variant of blocks world. The only way to achieve the goal in this domain is to move all the blocks to the table, and restack them. The problems in the blocks world do not require that all blocks be moved to the table.
- **TSP-1:** Traveling Sales Person domain where a salesman must visit a set of cities at least once. The operator here is  $drive(x,y)$  where  $x$  and  $y$  are cities.
- **TSP-2:** Traveling Sales Person domain where a salesman must visit a set of cities exactly once. The operator is still  $drive(x,y)$  as in TSP-1. However, there is a precondition  $unvisited(y)$  of  $drive(x,y)$  that is deleted by this operator. The domain contains non-invertible actions. An action is non-invertible if its effects cannot be reversed. Consider for example the action of driving from  $A$  to  $B$  under the constraint that we must not have visited  $B$  before. An add effect of the action is  $visited(B)$ . Such an action is non-invertible since the effect that  $B$  has been visited cannot be reversed. Specifically, there is no action which deletes  $visited(B)$ . Also, there is no action which adds  $unvisited(B)$ .
- **TSP-2n:** This is a traveling sales person domain where the salesman must visit a set of cities exactly once, except the relaxed cities. This domain is a variation of TSP-2. The operator in this domain is  $drive(x,y)$ . Only those cities mentioned in the goal need to be visited. This domain has dead ends. Relaxed cities are cities that can be visited multiple times. Relaxed cities always appear in the goal.
- **TSP-3:** Traveling Sales Person domain where a salesman can visit a set of cities either by flying or driving. The operators in this domain are  $drive(x,y)$ ,  $fly(x,y)$ ,  $get\_money(x)$ . The salesman can drive to a city at the most once. He/She can fly to any city an unlimited number of times, if he/she has enough cash. Cash can be found only at a small number of cities. This domain too contains non-invertible actions.

- **Package Routing:** There is a network, some nodes of which are connected by edges. Each connection allows a packet transfer in both directions. Only one packet can be transmitted over a connection at a time. Each packet has a start node and a destination node. Each connection has a capacity which is initially between 1 and 5, including 1 and 5. The capacity of a connection must be higher than zero for a package to be transmitted over it. Each packet transfer reduces the connection capacity by 1. These capacities are modeled using predicates, since numeric variables are not allowed in STRIPS. This domain has dead ends. Capacities of the connections never increase.
- **Modified Tower of Hanoi:** This is a variation of the well-known Tower of Hanoi domain. In the original domain, all discs need to be moved to one of the pegs such that only one disc is moved at a time and a larger disc is never put on a smaller disc. The modified domain allows any number of pegs and allows goals where there are discs at any number of pegs. This variation allows a wider variety of problems.
- **Gripper:** A robot has left gripper and right gripper. It can pick only one ball with one gripper. It can travel between rooms that are connected. The robot has to move balls to various rooms. This domain was used in the 1998 international planning competition.

The problems from the blocks world and block inversion domains were manually created. Problems from modified tower of Hanoi, TSP-1, TSP-2, TSP-2n and TSP-3 domains were automatically generated. The inter-city connections in the TSP problems were randomly generated. All problems we used have solutions. Each inter-city connection in TSP-1, TSP-2, TSP-2n and TSP-3 allows travel in both directions.

## 7.2. *A\**, *WA\**, and *AWA\**

Table 1 reports the performance of *A\**, *WA\** and *AWA\** on 8 problems from the blocks world. The problems range in size from 12 to 20 blocks. *AWA\** solved all 8 problems and *A\** and *WA\** solved only 3 and 4 problems respectively. *AWA\** was fastest on 7 out of the 8 problems.

Table 2 reports results of *A\**, *WA\** and *AWA\** on 8 problems from the TSP-1 domain. The problems range in size from 30 cities with at least 180 total inter-city connections to 35 cities with at least 210 total inter-city connections. *AWA\** solved all 8 problems. *A\** did not solve any problem. *WA\** solved 6 problems. The number of actions in plans found by *AWA\** were not higher than the number of actions in the plans found by *WA\** on 5 out of 6 problems that both these planners solved. *AWA\** was the fastest on 7 out of 8 problems.

### 7.3. AWA\*, FF 1.0 & STAN 4

Table 3 reports results of AWA\*, FF version 1.0, and STAN 4 on 9 problems from the block inversion domain. S in the last column of table 3 and tables 4 and 5 denotes speedup. Table 3 reports the speedup obtained with AWA\* over FF. AWA\* was the only planner to solve all 9 block inversion problems. FF and STAN 4 both solved 2 out of the 9 problems. AWA\* was fastest on 8 out of the 9 problems. We created 12 different instantiations of AWA\* and compared them with FF version 1.0 and STAN 4 on 6 problems from the block inversion domain. The instantiations of AWA\* were obtained by changing the values of the parameters in the tuple  $(\mu_h, \delta_h)$ . The instantiations had the following values: (1,2), (1,3), (2,1), (2,2), (2,3), (3,1), (3,2), (3,3), (4,1), (4,2), (4,3), and, (4,4). Out of the 12 instantiations of AWA\*, 1 solved 4 problems, 5 solved 5 problems and the remaining 6 solved all 6 problems. FF version 1.0 and STAN 4 both solved the same 2 problems from Table 3. All problems solved by FF and STAN 4 were solved by all 12 instantiations of AWA\*.

### 7.4. MAWA\*

Table 4 reports results obtained with AWA\* and MAWA\*. The first 8 problems are from the blocks world. The next 4 problems are from the TSP-2 domain. The last 4 problems are from the TSP-3 domain. The problems from blocks world range in size from 16 to 21 blocks. All problems from the TSP-2 and TSP-3 domains have 30 cities and at the most 100 inter-city connections. In the problems in the

Table 1. A\*, WA\*, AWA\* on Blocks World

Problem	A*		WA*		AWA*	
	acts	t	acts	t	acts	t
1	15	20.1	18	15.5	16	7.05
2	23	214.1	29	60.4	30	50.3
3	-	> 600	-	> 600	20	39.1
4	-	> 600	-	> 600	40	404.5
5	-	> 600	-	> 600	16	10.3
6	-	> 600	28	335.8	27	336.1
7	-	> 600	-	> 600	16	18.5
8	18	109.5	15	45.1	15	45.5

Table 2. A\*, WA\*, AWA\* on TSP-1

Problem	A*		WA*		AWA*	
	acts	t	acts	t	acts	t
1	-	> 600	28	5.1	28	4.7
2	-	> 600	25	31.5	25	3.1
3	-	> 600	28	23.6	27	3.6
4	-	> 600	30	143.7	29	23
5	-	> 600	-	> 600	36	420.9
6	-	> 600	39	335.8	38	354.2
7	-	> 600	-	> 600	36	331.8
8	-	> 600	36	453.1	39	363.7

Table 3. AWA\*, FF version 1.0, STAN 4 on Blocks inversion

# Blocks	AWA*		FF		STAN 4		S
	acts	t	acts	t	acts	t	
10	23	3.1	17	1.3	13/21	1.1	-
11	23	3.9	19	5.1	15/23	9.4	1.3
12	29	12.6	-	> 600	-	> 600	> 47
13	35	33.5	-	> 600	-	> 600	> 17
14	35	45.7	-	> 600	-	> 600	> 13
15	41	111.7	-	> 600	-	> 600	> 5.3
16	41	138.0	-	> 600	-	> 600	> 4.3
17	47	304.2	-	> 600	-	> 600	> 1.9
18	47	394.4	-	> 600	-	> 600	> 1.5

TSP-3 domain, the cash is located in at the most 6 cities. The actual cash locations are randomly generated. We also report the speedup obtained with MAWA\* over AWA\*. MAWA\* solved all 16 problems. AWA\* solved 11 problems. MAWA\* was fastest on 15 problems. On 6 out of the 11 problems that both MAWA\* and AWA\* solved, the plans found by MAWA\* had far fewer actions than the plans found by AWA\*.

Table 5 reports results on the TSP-3 domain. The problems range in size from 30 cities with at the most 100 connections and at the most 6 cash locations to 40 cities with at the most 140 connections and at the most 8 cash locations. We also ran AltAlt on these problems, however, AltAlt gave a memory allocation error after around 4 cpu minutes during the search. The memory allocation error occurred before AltAlt could find solution to any of the problems. The speedup obtained by MAWA\* over FF is also reported for this domain. In the TSP-3 domain, 15 problems were solved by MAWA\* and 10 were solved by FF. All problems solved by FF were also solved by MAWA\*. MAWA\* also found much shorter plans for all of the problems solved by FF. The implementation of MAWA\* used fixed  $\epsilon = 0.2$ . We also tried MAWA\* with  $\epsilon = 0.1$ ,  $\epsilon = 0.3$ , and variable  $\epsilon$ . The implementation which used variable  $\epsilon$  continuously changed  $\epsilon$  after every 15 nodes were expanded. The performances with  $\epsilon = 0.1$ ,  $\epsilon = 0.3$  and with variable  $\epsilon$  were on par with that of MAWA\* with fixed  $\epsilon = 0.2$  on most problems.

It is hard to empirically compare optimality of A\* and AWA\*, since AWA\* solved many more problems than A\*. AWA\* and MAWA\* found shorter plans than FF on many problems.

### 7.5. Additional Results with AWA\*, FF 1.0 & STAN 4

We report on evaluation on 16 additional problems in this subsection. Table 6 reports the performance of AWA\*, FF version 1.0, and STAN 4 on 16 problems. The first 8 problems are from blocks world, and the remaining 8 problems are from the TSP-1 domain. The blocks world problems range in size from 14 to 20 blocks. The problems from the TSP-1 domain range in size from 30 cities with at least 180 connections to 35 cities with at least 210 connections. The actual connections

Table 4. AWA\*, MAWA\* on Blocks World, TSP-2 and TSP-3

Problem	AWA*		MAWA*		S
	acts	t	acts	t	
1	26	60.3	16	29.3	2
2	42	225.3	14	10.8	20.8
3	23	392.6	17	68.2	5.7
4	-	> 600	20	157.3	> 3.8
5	31	455.7	30	240.8	1.9
6	14	46.1	14	50.5	-
7	22	101.9	20	96.8	-
8	-	> 600	33	251.8	> 2.4
9	27	46.8	25	7.3	6.4
10	31	62.8	32	9.6	6.5
11	-	> 600	33	29.9	> 20
12	31	211	32	8.4	25
13	-	> 600	36	40.9	> 14.6
14	33	231.6	36	32.8	7
15	34	368.7	39	42.9	8.7
16	-	> 600	45	36.6	> 16.4

Table 5. MAWA\* and FF version 2.3 on TSP-3

Problem	MAWA*		FF		S
	acts	t	acts	t	
1	30	3.8	40	0.4	-
2	25	4.5	38	6.4	1.4
3	27	5.3	34	9.1	1.7
4	-	> 600	-	> 600	-
5	-	> 600	-	> 600	-
6	32	16.1	43	3	-
7	37	28.1	49	22.1	-
8	35	32.4	43	26.5	-
9	30	43.5	45	31.4	-
10	33	44.7	41	43.9	-
11	34	47.8	-	> 600	> 12.5
12	35	56.8	-	> 600	> 10.5
13	37	58.9	-	> 600	> 10.2
14	-	> 600	-	> 600	-
15	-	> 600	-	> 600	-
16	35	9.4	50	3.6	-
17	41	73.8	46	10.5	-
18	39	90.2	-	> 600	> 6.6
19	41	118.6	-	> 600	> 5
20	-	> 600	-	> 600	-

are randomly generated. AWA\* is the only planner that solved all 16 problems. FF solved 14 problems. STAN 4 solved 2 problems. FF was fastest on all problems that it solved. On 12 out of the 14 problems that both FF and AWA\* solved, plans found by AWA\* had far fewer actions than those in the plans found by FF. On the two problems that both AWA\* and STAN 4 solved, plans found by AWA\* had

fewer actions than those in the plans found by STAN 4. Evaluation on many other problems from the five domains confirms the superiority of AWA\* and MAWA\*. This evaluation is reported in our longer technical report.<sup>20</sup>

Table 6. AWA\*, FF version 1.0, STAN 4 on Blocks World and TSP-1

Problem	AWA*		FF		STAN 4	
	acts	t	acts	t	acts	t
1	16	10.5	16	1.1	9/22	520.6
2	30	70.4	-	> 600	-	> 600
3	20	59.6	21	2.8	-	> 600
4	40	581.7	20	3.1	-	> 600
5	15	16.3	19	0.9	-	> 600
6	27	472.1	-	> 600	-	> 600
7	16	30.7	20	2.2	-	> 600
8	15	68.5	17	2.3	6/26	106.5
9	23	27.9	30	0.2	-	> 600
10	26	35.7	35	0.3	-	> 600
11	23	29.2	38	0.3	-	> 600
12	21	23.1	29	0.2	-	> 600
13	38	420.9	55	3.1	-	> 600
14	36	354.2	50	2.7	-	> 600
15	38	331.8	51	2.5	-	> 600
16	39	363.7	52	2.7	-	> 600

## 7.6. AWA\*-AC & AWA\*-PD

Table 7. AWA\*-AC and HSP-2 on TSP-3

Problem	AWA*-AC		HSP-2	
	acts	t	acts	t
tsp3-1	19	2.91	19	0.31
tsp3-2	26	47.4	-	-
tsp3-3	27	48.6	-	-
tsp3-4	31	34.6	34	100.04
tsp3-5	21	24.53	24	0.25
tsp3-6	28	106.7	-	-
tsp3-7	32	86.2	35	396.5
tsp3-8	22	378.2	-	-
tsp3-9	26	41.2	24	252.1
tsp3-10	39	14.7	43	1.5

In this subsection, we report on an empirical evaluation of AWA\*-AC and AWA\*-PD. In Table 7, - means that no plan was found in 10 cpu minutes. In the 10 problems from Table 7, the number of cities in the initial state varied from 30 to 40. The number of cities in goal varied from 27 to 36. The number of inter-city connections varied from 90 to 120. Table 7 shows that AWA\*-AC solved all problems and HSP-2 solved 6 problems.

Table 8. AWA\*-AC and AWA\*-PD on Blocks World

Problem	AWA*-AC		AWA*-PD	
	acts	t	acts	t
bw-11	19	76.3	19	80.7
bw-12	47	338.8	51	354
bw-13	36	72.8	36	74.4
bw-14	40	185.2	40	183.4
bw-15	11	170.6	11	170.5
bw-16	15	451.5	15	463.4

As Table 8 shows, performance of AWA\*-AC and AWA\*-PD does not differ significantly on blocks world problems. Table 9 shows that there is a significant difference in the performance of the two planners on TSP-3 domain. In fact AWA\*-PD solved none of the 8 problems from Table 9. In Table 9, - means that the problem was not solved in 20 cpu minutes.

Table 9. AWA\*-AC and AWA\*-PD on TSP-3

Problem	AWA*-AC		AWA*-PD	
	acts	t	acts	t
tsp3-12	30	97.5	-	-
tsp3-13	28	19.9	-	-
tsp3-15	27	21	-	-
tsp3-16	27	33.8	-	-
tsp3-18	31	15.3	-	-
tsp3-19	27	78.6	-	-
tsp3-20	29	10.2	-	-
tsp3-29	35	292.3	-	-

A comparison of AWA\*-AC, HSP-2 and FF on some problems from TSP-3 is reported in Table 10. In Table 10, \* means that no plan was found in 20 cpu minutes. In the problems in Table 10, the number of cities in initial states was 30, the number of inter-city connections varied from 80 to 90 and the number of cash locations was 5. The results in Table 10 show that AWA\*-AC solved some problems that HSP-2 or FF did not.

Performance of AWA\*-AC, HSP-2 and FF on some problems from TSP-2n is reported in Table 11. In Table 11, \* denotes that the problem was not solved in 20 cpu minutes. The number of cities in initial states of the problems in Table 11 varied from 30 to 35. The number of relaxed cities varied from 6 to 8. The number of cities in the goal of these problems varied from 25 to 28. The number of inter-city connections in the first 10 problems (tsp2n-1 ... tsp2n-10) varied from 80 to 90. The number of inter-city connections in the last 10 problems (tsp2n-11 ... tsp2n-20) varied from 90 to 105. AWA\*-AC did solve some problems that FF or HSP-2 did not. Whenever FF solved a problem, it was almost always the fastest planner. AWA\*-AC was faster than HSP-2 on some of the problems that both these planners

Table 10. AWA\*-AC, HSP-2 and FF on TSP-3

Prob	AWA*-AC		HSP-2		FF	
	acts	t	acts	t	acts	t
tsp3-21	38	580.9	42	14.3	40	41.3
tsp3-22	*	*	39	8.8	42	14.7
tsp3-23	*	*	*	*	*	*
tsp3-24	37	192.3	41	857.6	42	11.4
tsp3-25	*	*	*	*	41	7.4
tsp3-26	*	*	41	279.3	39	6.4
tsp3-27	*	*	*	*	*	*
tsp3-28	*	*	*	*	*	*
tsp3-29	35	292.3	*	*	*	*
tsp3-30	*	*	*	*	*	*
tsp3-31	*	*	*	*	*	*
tsp3-32	30	97.5	28	14.2	33	14.7
tsp3-33	33	19.9	*	*	*	*
tsp3-34	*	*	*	*	35	16.1
tsp3-35	29	20.9	*	*	*	*
tsp3-36	32	33.8	31	12.7	*	*
tsp3-37	32	179.6	35	33.1	34	4.1
tsp3-38	31	15.3	35	8.9	35	3.7
tsp3-39	30	78.6	*	*	31	8.3
tsp3-40	29	10.2	*	*	*	*

solved. tsp2n-11 is the only problem that both HSP-2 and FF solved and AWA\*-AC did not. Tables 7,10 and 11 show that the number of actions in the plans found by AWA\*-AC is lower than the number of actions in the plans of HSP-2 and/or FF on many problems.

### 7.7. AWA\*-AC-LE

The results with AWA\*-AC-LE and AWA\*-AC are reported in Table 12. AWA\*-AC-LE used  $p = 6$ ,  $\theta_1 = 3$ ,  $\theta_2 = 5$ , and  $\alpha = 1.5$ . We compared AWA\*-AC-LE and AWA\*-AC on 30 problems from TSP-3 domain, all of which are mentioned in Table 12. AWA\*-AC-LE solved 26 problems and AWA\*-AC solved 22 problems. AWA\*-AC-LE solved 23 problems faster than AWA\*-AC. AWA\*-AC-LE solved 3 problems at least 100 times faster than AWA\*-AC. AWA\*-AC-LE solved many problems at least 10 times faster than AWA\*-AC. In Table 12, - denotes that the problem was not solved in 20 cpu minutes.

AWA\*, MAWA\*, AWA\*-AC, AWA\*-PD and AWA\*-AC-LE all use some parameters like  $\mu_g, \mu_h, c$  etc. We did not tune these parameters to obtain the empirical results. We did evaluate the variants on several domains from the international planning competitions in 1998, 2000 and 2002. The variants solved many problems from these domains, but they were not as efficient as FF and HSP-2. HSP-2 is much faster than our planners in domains like ZenoTravel, Rovers, Mystery and Gripper. We did run AltAlt<sup>3</sup> and STAN 4<sup>15</sup> on some problems from TSP domains. The variants of A\* were more efficient than these planners on these problems.

Table 11. AWA\*-AC, HSP-2 and FF on TSP-2n

Problem	AWA*-AC		HSP-2		FF	
	acts	t	acts	t	acts	t
tsp2n-1	27	24.6	29	19.4	28	2.1
tsp2n-2	28	319.9	*	*	*	*
tsp2n-3	30	117.9	28	25.9	29	7.3
tsp2n-4	27	35.9	*	*	*	*
tsp2n-5	30	149.6	29	333.7	*	*
tsp2n-6	27	380.9	26	12.4	30	2.8
tsp2n-7	26	1	27	9.35	28	1.1
tsp2n-8	28	156.5	29	8	29	8.3
tsp2n-9	29	4.7	29	6.2	27	0.7
tsp2n-10	28	2.6	29	5.3	29	0.5
tsp2n-11	*	*	32	12.9	32	11.4
tsp2n-12	32	79.9	34	28.8	*	*
tsp2n-13	*	*	*	*	*	*
tsp2n-14	*	*	*	*	*	*
tsp2n-15	35	74.7	34	42.9	*	*
tsp2n-16	32	184.4	34	500.4	32	2.2
tsp2n-17	*	*	*	*	*	*
tsp2n-18	*	*	*	*	*	*
tsp2n-19	32	175.9	32	33.4	*	*
tsp2n-20	30	36.6	32	80.7	33	3.9

### 7.8. Varying the Parameter Values

The variants require values of several parameters. Is a careful selection of parameters necessary to obtain a superior performance from these planners? This motivated a study of these planners with different values of the parameters. We wanted to test the robustness of these planners to changes in the values of the parameters. We ran the five variants with multiple parameter value sets on 70 problems from five domains. 15 problems from Package routing were used. 15 problems from Modified Tower of Hanoi were used. 15 problems from TSP-2 were used. 15 problems from TSP-3 were used. 10 problems from blocks world were used. AWA\* was run with five parameter value sets. MAWA\* was run with eleven parameter value sets. AWA\*-AC was run with eight parameter value sets. AWA\*-PD was run with eight parameter value sets. AWA\*-AC-LE was run with twenty five parameter value sets. So a total of fifty seven parameter sets were used. Since 70 problems were used, there were 3990 problem solving attempts ( $70 * 57$ ). This evaluation was conducted on a SunBlade workstation with Linux operating system. The cutoff was 30 cpu minutes. For each run of each planner on each domain, we found the number of problems solved, the total solving time (TST) and the average solving time (AST).

The parameter value sets used in five runs of AWA\* were as follows:  $\langle \mu_g = 0.4, \mu_h = 2 \rangle$ ,  $\langle \mu_g = 0.3, \mu_h = 2 \rangle$ ,  $\langle \mu_g = 0.2, \mu_h = 2 \rangle$ ,  $\langle \mu_g = 0.8, \mu_h = 2 \rangle$ , and  $\langle \mu_g = 0.5, \mu_h = 7 \rangle$ .  $\delta_h$  was 1 in all runs. All five runs of AWA\* solved the same number of problems from each domain. The TST and AST values of all runs differed by at the most around 10 % on each domain. This shows that changes in

Table 12. AWA\*-AC-LE and AWA\*-AC on TSP-3

Problem	AWA*-AC-LE		AWA*-AC	
	acts	t	acts	t
tsp-41	28	33.8	26	75.6
tsp-42	26	32.2	26	16.6
tsp3-43	27	11	-	-
tsp3-44	25	8.9	27	126.4
tsp-45	26	20.7	25	19.1
tsp-46	29	5.3	27	32.1
tsp-47	28	4.2	28	17.4
tsp3-48	27	92	27	1159.8
tsp-49	29	11.6	28	127.9
tsp-50	35	230.3	-	-
tsp3-51	32	8.1	29	611.1
tsp3-52	31	16.3	32	777.5
tsp-53	28	12.44	28	189.4
tsp-54	-	-	30	374.3
tsp-55	31	602.2	-	-
tsp-56	28	5.3	29	28.4
tsp-57	32	47.1	33	81.2
tsp3-58	31	328.6	31	93
tsp-59	31	37.1	29	72.1
tsp-60	28	10.4	30	113.7
tsp3-61	-	-	35	216.3
tsp3-62	35	8.8	-	-
tsp-63	-	-	-	-
tsp-64	36	9.6	34	56
tsp-65	35	72.6	33	1021.4
tsp-66	32	27.5	30	490.5
tsp3-67	35	15.5	-	-
tsp-68	-	-	-	-
tsp-69	37	72.4	33	227.4
tsp3-70	33	8.2	-	-

the parameters' values did not significantly affect the performance of AWA\*.

MAWA\* was run with eleven parameter value sets. All eleven runs used  $\mu_g = 0.5, \delta_h = 1$  and  $\mu_h = 2$ . The values of  $\epsilon$  and  $c$  used by these runs in the format  $\langle \epsilon, c \rangle$  are as follows:  $\langle 0.2, 13 \rangle, \langle 0.2, 16 \rangle, \langle 0.2, 20 \rangle, \langle 0.2, 25 \rangle, \langle 0.15, 10 \rangle, \langle 0.25, 10 \rangle, \langle 0.35, 10 \rangle, \langle 0.45, 10 \rangle, \langle 0.1, 5 \rangle, \langle 0.2, 9 \rangle$ , and  $\langle 0.3, 13 \rangle$ . All eleven runs solved the same number of problems from each domain. The TST values of these runs differed by at the most around 10 % on each domain. For most runs, the difference was at the most 5 %. The AST values of these runs too differed by at the most around 10 %. This shows that MAWA\* is robust to the changes in the the parameter values.

The values of  $\mu_h$  and  $c$  used by 8 runs of AWA\*-AC in  $\langle \mu_h, c \rangle$  format are as follows:  $\langle 2, 5 \rangle, \langle 2, 12 \rangle, \langle 2, 17 \rangle, \langle 2, 22 \rangle, \langle 3, 5 \rangle, \langle 5, 9 \rangle, \langle 7, 13 \rangle$ , and  $\langle 9, 17 \rangle$ . All of these runs used  $\mu_g = 0.5, \delta_h = 1$ . The number of problems from any domain solved by different runs differed by at the most 1. The TST values of

different runs on any domain differed by at the most around 10 %. The AST values of the eight runs too differed by at the most around 10 % on any domain. This again shows that AWA\*-AC is robust to the parameter changes.

The values of all parameters used by the 8 runs of AWA\*-PD were exactly same as those used by the 8 runs of AWA\*-AC. The number of problems solved by different runs of AWA\*-PD on any domain differed by at the most 2. The TST values of the runs differed by at the most 4 % on Package routing, TSP-3 and TSP-2. The TST values of runs using  $c = 5, 12,$  and  $22$  were within 1 % of each other on Modified Tower of Hanoi. The TST values of runs using  $c = 9, 13,$  and  $17$  were around twice the TST values of runs using  $c = 5, 12,$  and  $22$  on Modified Tower of Hanoi. The AST values of runs using  $c = 5, 12,$  and  $22$  were within 0.5 % of each other on Modified Tower of Hanoi. On blocks world, the TST and AST values of the runs using  $\mu_h = 2$  differed by around 0.01 %. On this domain, the TST and AST values of runs using  $\mu_h = 3, 5, 7$  and  $9$  were respectively around 30 % and around 20 % less than the TST and AST values of runs using  $\mu_h = 2$ .

The values of the parameters used by 25 runs of AWA\*-AC-LE in  $\langle p, \theta_1, \theta_2, \alpha \rangle$  format are as follows:  $\langle 2, 3, 5, 1.5 \rangle,$   $\langle 4, 3, 5, 1.5 \rangle,$   $\langle 8, 3, 5, 1.5 \rangle,$   $\langle 10, 3, 5, 1.5 \rangle,$   $\langle 6, 3, 5, 3 \rangle,$   $\langle 6, 3, 5, 4 \rangle,$   $\langle 6, 3, 5, 5 \rangle,$   $\langle 6, 3, 5, 6 \rangle,$   $\langle 6, 4, 5, 1.5 \rangle,$   $\langle 6, 6, 5, 1.5 \rangle,$   $\langle 6, 8, 5, 1.5 \rangle,$   $\langle 6, 10, 5, 1.5 \rangle,$   $\langle 6, 3, 7, 1.5 \rangle,$   $\langle 6, 3, 9, 1.5 \rangle,$   $\langle 6, 3, 11, 1.5 \rangle,$   $\langle 6, 3, 13, 1.5 \rangle,$   $\langle 3, 3, 5, 2 \rangle,$   $\langle 5, 3, 5, 4 \rangle,$   $\langle 8, 3, 5, 6 \rangle,$   $\langle 6, 4, 6, 1.5 \rangle,$   $\langle 6, 8, 8, 1.5 \rangle,$   $\langle 6, 12, 10, 1.5 \rangle,$   $\langle 15, 3, 5, 1.5 \rangle,$   $\langle 10, 3, 5, 1.5 \rangle,$  and  $\langle 8, 3, 5, 1.5 \rangle$ . Since this variant uses more parameters than other variants, we had more runs of this variant. All 25 runs used  $\mu_g = 0.5,$   $\delta_h = 1,$  and  $\mu_h = 2$ . First 22 runs used  $c = 10$ . 23rd, 24th and 25th runs respectively used  $c = 7, 12$  and  $15$ . The number of problems solved by the 25 runs on any domain differed by at the most 3. There was a much wider variation in the AST and TST values of the runs on TSP-2, TSP-3, modified tower of Hanoi and blocks world.

A comparison of the five variants using total solving time and the number of problems solved is reported in table 13. MTOH in this table denotes ‘‘Modified Tower of Hanoi’’. The table shows the maximum number of problems solved by each planner from each domain, considering all of its runs. The table also shows the minimum of the total solving times of such runs in parantheses. The total solving times are based only on solved problems. For example, 4 runs out of 8 runs of AWA\*-PD all solved 8 out of the 10 problems from blocks world and the time reported in the table is  $\min(4184.01, 4184.4, 4184.28, 4184.16)$ . The last row of the table shows the total number of problems that each planner solved, out of the 70 problems attempted. The number of solved problems reported in the last row is a summation of the number of solved problems reported in the same table. The results clearly show that node-dependent weighting alone performs very poorly as compared with other variants. The results also show that conditional two-phase heuristic evaluation is a good idea. The results also show that lazy evaluation is a good idea. Overall, AWA\*-AC-LE was superior. No run of any other planner on

any domain solved more problems than the maximum number of problems solved by the runs of AWA\*-AC-LE. This is also clear from table 13.

Table 13. Comparison of five variants on 70 problems

Domain	AWA*	MAWA*	AWA*-AC	AWA*-PD	AWA*-AC-LE
Package Routing	9	11	11	11	11
	(843.92)	(789.89)	(777.77)	(762.31)	(762.31)
TSP-3	0	4	11	11	12
		(1310.35)	(2008.25)	(2247.88)	(1001.45)
TSP-2	11	11	10	10	11
	(702.67)	(839.41)	(69.98)	(69.82)	(240.85)
MTOH	5	10	11	9	12
	(402.51)	(2135.1)	(1674.74)	(1509.07)	(2775.7)
Blocks World	1	7	8	8	8
	(2.61)	(2622.68)	(1636.48)	(4184.01)	(1941.58)
Total	26 / 70	43 / 70	51 / 70	49 / 70	54 / 70

### 7.9. AWA\* and WA\*

In this subsection, we report on an empirical comparison of AWA\* with WA\* using six different values of  $w$  in the path cost equation  $f(n) = g(n) + w * h(n)$ . The goal of this evaluation was to determine if node-dependent weighting is better than static weighting for state-space STRIPS planning. WA\* uses static weighting. The evaluation was conducted on a SunBlade workstation with Linux operating system. The cutoff used was 30 cpu minutes. We ran WA\* with six values of  $w$  (2,3,4,5,6 and 7) on 6 domains. We also ran AWA\* on these domains. The domains are package routing, TSP-3, modified tower of Hanoi, TSP-2, blocks world and gripper. Ten problems from each domain were used. The outcome of the evaluation on the 60 problems is reported in Table 14. S in the second column of this table denotes the number of problems solved. LA in the third column of this table denotes the number of problems solved by 2 or more of the 7 planners where the planner found plan with lowest number of actions among the plans found by various planners. PR in the fourth column denotes Package Routing domain. MHT in the sixth column denotes Modified Tower of Hanoi domain. The last 6 columns of Table 14 show the number of problems solved by AWA\* and WA\* from different domains listed in the table.

Table 14. A Comparison of AWA\* and WA\*

Planner	S	LA	PR	TSP-3	MHT	TSP-2	Blocks	Gripper
AWA*	46	24	3	9	10	8	8	8
WA*, $w = 2$	43	33	3	8	10	9	8	5
WA*, $w = 3$	44	17	3	9	10	8	8	6
WA*, $w = 4$	46	18	3	9	10	8	8	8
WA*, $w = 5$	47	19	3	9	10	8	9	8
WA*, $w = 6$	48	19	3	10	10	8	9	8
WA*, $w = 7$	48	19	3	10	10	8	9	8

The solving times of AWA\* ranged from 0.19 seconds to 1693.88 seconds. Solving times of WA\* with  $w = 2$  ranged from 0.18 seconds to 1374.16 seconds. Solving times of WA\* with  $w = 3$  ranged from 0.19 seconds to 1626.2 seconds. The solving times of WA\* with  $w = 4$  ranged from 0.19 seconds to 1295.25 seconds. The solving times of WA\* with  $w = 5$  ranged from 0.19 seconds to 1566.93 seconds. Solving times of WA\* with  $w = 6$  ranged from 0.2 seconds to 936.16 seconds. Solving times of WA\* with  $w = 7$  ranged from 0.18 seconds to 936.26 seconds. AWA\* solved more problems than WA\* with  $w = 2$  and  $w = 3$ . WA\* with  $w = 4, 5, 6$  and  $7$  solved as many or more problems than AWA\*. However the plans found by WA\* with  $w = 4, 5, 6,$  and  $7$  had more actions than the plans found by AWA\* on many problems. So node-dependent weighting appears to be useful if both the number of problems solved and plan optimality are important for a user.

## 8. Discussion

In this paper, we reported on five sound and complete classical planners AWA\*, MAWA\*, AWA\*-AC, AWA\*-PD and AWA\*-AC-LE. AWA\* uses node-dependent weights. MAWA\*, AWA\*-AC, AWA\*-PD and AWA\*-AC-LE use conditional two-phase heuristic evaluation along with node-dependent weights. AWA\*-AC-LE also uses lazy evaluation. We reported on the empirical performance of these planners. In this section, we discuss related work, directions for future work, along with our theoretical work on additional variants of A\*.

### 8.1. Related Work

Heuristic path algorithm (HPA)<sup>7</sup> is very similar to Weighted A\*. HPA uses the cost equation  $f(x) = (1 - w) * g(x) + w * h(x)$ ,  $0 \leq w \leq 1$ . Pohl mentions that the evaluation function can be extended to a general linear form or convex combination  $f(x) = (1 - \sum_{i=1}^k w_i) * g(x) + (\sum_{i=1}^k (w_i * h_i(x)))$ ,  $w_i \geq 0$ ,  $(\sum_{i=1}^k w_i) \leq 1$ . Pohl does mention that  $w_i$  could be functions on the nodes, however no experimental results are presented on any of the search algorithms mentioned in the paper.

KWA\*<sup>21</sup> is a variant of WA\* that expands  $K \geq 1$  nodes with smallest  $f$  values per iteration.  $K$  is a user-specified parameter. This gives diversity to WA\*. MSC-WA\* (Multi-state commitment WA\*)<sup>22</sup> focuses search on a subset of the nodes in the fringe. The size of this subset is determined by user-specified parameter  $C$ . MSC-KWA\*<sup>23</sup> adds both diversity and commitment to WA\*. MSC-KWA\* is shown to be superior to WA\*, KWA\* and MSC-WA\* on 35-puzzle, 48-puzzle and 4-peg Tower of Hanoi problems.

Beam search expands nodes in a breadth-first style. It expands only  $w$  most-promising nodes at each level and removes others permanently.  $w$ , a user-specified parameter is called beam width. BSIDA\*<sup>24</sup> is Beam-Stack Iterative-Deepening A\*. This is a variation of beam search which is complete and whose space complexity is linear in the depth of the search. BSIDA\* has been shown to outperform depth-first

iterative-deepening A\* (DFIDA\*) on STRIPS planning problems. Limited discrepancy search (LDS)<sup>25</sup> is designed to work on finite binary trees. Children of a node are sorted in the order of increasing heuristic values. So the heuristic values always show the left child as being preferable to the right child. Choosing the right child for expansion is called discrepancy. LDS first searches without any discrepancy. Then it searches the tree with an increase in the number of discrepancies allowed. BULB<sup>26</sup> is beam search using limited discrepancy backtracking. BULB has been shown to outperform beam search and variants of WA\*. Hybrid AcE<sup>27</sup> is a domain-independent planning system that combines forward chaining and backward chaining. The forward search occurs in WA\* style until the heuristic function is no longer capable of guiding the search. Then backward search is invoked. The backward search tries to reach the best state in the fringe of the forward search. The search direction may change many times before solution is found. LPG<sup>28</sup> is a highly-efficient planner using local search. It uses a linear weighted function with three parameters to estimate the cost of adding an action to the existing partial plan. It uses a similar function to estimate the cost of removing an action from the existing partial plan. LPG is incomplete. LPG handles actions with time durations which our planners in this paper do not.

An approach to dynamic weighting that uses estimates of the error in heuristic values is presented by Koll & Kaindl.<sup>8</sup> It has been shown that this weighting scheme significantly reduces the number of nodes expanded on 8-puzzle problems. This dynamic weighting scheme involves computation of error  $h^*(n) - h(n)$  for all nodes  $n$  in a sample.  $h^*(n)$  is the cost of an optimal path from node  $n$  to goal. A\* needs to be used to find  $h^*(n)$  for each node in the sample unless there is a computationally easier approach. The errors for nodes in the sample are used to estimate the errors for all nodes and the error estimates for the nodes are used in the weighting function for  $h(n)$  in the equation for  $f(n)$ . Applying this approach to planning will mean solving many planning problems with A\* before solving the given problem. This approach involving learning a problem-dependent heuristic error function is likely to considerably increase the solving time. Solving times with and without dynamic weighting are not reported.<sup>8</sup> *HAPRC* is an adaptive planning system which uses machine learning.<sup>29</sup> Data about problem attributes, values of the planning parameters and performance (good or bad) is fed into a machine learning tool to learn rules associating good performance with planning parameters and problem attributes. The rules are transformed into production rules that tune planner parameters in the presence of certain problem characteristics. The rules are embedded into a planner. Such an approach can be applied to our planners to learn to predict the good values of various parameters. Each of our parameters is intended to integrate some node-evaluation criterion with the original A\*. The more the criteria used, the higher is the number of parameters. This makes the use of machine learning for learning parameter-predicting rules for our variants a worthwhile activity. There are many more variants of A\*, e.g. Differential A\*<sup>30</sup> is

a variant that handles simultaneous changes in graph topology, transition costs or start/goal. It has been applied to robot path planning.

## 8.2. Insights based on waiting times in node expansions

We believe that some of the nodes on the way to goal were not expanded at all or not expanded soon, on the problems where our planners took a long time to solve or did not solve within the time cutoff. To test this intuition, we found waiting times for nodes in separate experiments conducted after the empirical evaluation in section 7. The waiting time of a node  $n$  is the iteration difference ( $et_2(n) - et_1(n)$ ), where  $et_i(n)$  denotes the iteration on which  $n$  is expanded when heuristic evaluation with  $i$  phases is used.  $et_2(n)$  is the iteration on which node  $n$  is expanded when conditional two-phase heuristic evaluation is used.  $et_1(n)$  is the iteration on which  $n$  is expanded when just the relaxed plan heuristic as in FF is used (and adjustment from second phase is not applied). The waiting times were found in our implementation as follows. When the next node to be expanded was to be found on  $i$  th iteration, node  $n$  with minimum  $f$  value based on the cost equation  $f(n) = g(n) + h_1(n)$  was found.  $h_1$  is the heuristic used in the first phase of the conditional two-phase heuristic evaluation. Then the node was added to a waiting queue and  $et_1(n)$  was set to  $i$ . Note that  $g(n) + h_1(n)$  values are used only for finding the iterations at which nodes would have been expanded in absence of multi-phase heuristic evaluation. These values are not used in actually expanding a node. The entry time of a node is the iteration on which it would have been expanded, had only  $h_1$  been used. Then node  $n'$  with minimum  $f$  value based on the cost  $f(n') = g(n') + h_2(n')$  was found.  $h_2(n') = h_1(n')$  if  $h_1(n') \geq c$ , where  $c$  is the user-specified cutoff used to decide whether to apply the second heuristic evaluation phase, as described in the algorithms of MAWA\*, AWA\*, AWA\*-AC, AWA\*-AC-LE and AWA\*-PD. If  $n'$  was found in the waiting queue,  $n'$  was removed from the waiting queue and  $et_2(n')$  was set to  $i$ , the iteration on which  $n'$  was actually expanded. The waiting time of a node is the iteration on which it is expanded when conditional two-phase heuristic evaluation is used minus the iteration on which it is expanded when just one-phase heuristic evaluation is used. If the waiting time of a node is high, it means that the two-phase evaluation delays the expansion of the node a lot.

There are six cases involving  $et_1(n)$  and  $et_2(n)$  that we describe next to clarify the notion of waiting time. (i) It is possible for a node not to have any of the  $et_1()$  and  $et_2()$  values. (ii) It is possible for  $et_1(n)$  and  $et_2(n)$  to be equal. (iii) It is not possible to have  $et_1(n) > et_2(n)$ . (iv)  $et_1(n) < et_2(n)$  is possible. (v) It is possible for a node  $n$  to have  $et_2(n)$  value and not have  $et_1(n)$  value. (vi) It is possible for a node  $n$  to have  $et_1(n)$  value and not have  $et_2(n)$  value. AWA\*-AC was extended to find waiting times of nodes. Waiting times of nodes were found on several of the difficult problems from the TSP-2, TSP-3, Zeno Travel domains. The waiting times were negligible for most of the nodes. Still there were some nodes with waiting

time in excess of 30, and there were a small number of nodes with waiting times in exceeding 100. There were a few nodes with waiting times exceeding 500. Some of these nodes with very high waiting times belonged to the plans found by the planner. High waiting times of these nodes result from an over-adjustment of their  $h$  values in the second phase. This explains why AWA\*-AC is slow on several problems. One potential modification to handle this is to limit the waiting time to avoid a highly delayed expansion of relevant nodes. If  $et_1(n) = 20$  and  $n$  is still unexpanded after 400 iterations, then the waiting time of  $n$  is higher than 380. Such lower bounds on the waiting times for unexpanded nodes that have  $et_1(n)$  value can be continuously updated and all nodes that have waited for more than  $\beta$  iterations can be expanded, whether their  $f$  values (based on  $g$  and  $h_2$ ) are lowest or not.  $\beta$  is a user-specified parameter. Using  $\beta$  limits the waiting times of nodes. Investigating the utility of limiting the waiting time in improving the performance of our variants is our future work.

### 8.3. Additional Variants of A\*

Our conditional two-phase heuristic evaluation can be viewed as a special case of conditional  $n$ -phase heuristic evaluation, where  $i$  th phase re-evaluates far fewer nodes than  $(i - 1)$  th phase.  $c$  and  $\epsilon$  can both be varied during search. We report on 9 variants of A\* in another paper.<sup>18</sup> One variant there is a variant of MAWA\* which uses conditional  $n$ -phase heuristic evaluation, where  $n$  may be greater than 2. One variant uses  $n$ -phase heuristic evaluation along with variable  $\epsilon$ . One variant changes heuristics during search.<sup>18</sup> Change of heuristics during search is different from  $n$ -phase heuristic evaluation. One variant changes heuristics during search and also varies  $\epsilon$ . We have reported on the upper bounds on the costs of solutions found by the nine variants of A\*,<sup>18</sup> assuming that the heuristics are inadmissible. Empirically evaluating these algorithms on planning problems is our future work.

## 9. Conclusion

We reported on five variants of the A\* algorithm for planning. All variants (AWA\*, MAWA\*, AWA\*-AC, AWA\*-PD and AWA\*-AC-LE) are sound and complete. AWA\* uses node-dependent weights to adjust the heuristic estimates. MAWA\*, AWA\*-AC, AWA\*-PD and AWA\*-AC-LE use conditional two-phase heuristic evaluation. They use a computationally cheap heuristic in phase 1 and adjust the heuristic estimates of some nodes in phase 2, using a different heuristic. The conditional two-phase evaluation allows a use of more heuristic information without significantly increasing the computations. MAWA\* uses node-dependent weights for only a selected set of nodes in the fringe. AWA\*-AC-LE uses lazy evaluation to save the time spent on constructing relaxed planning graphs and extracting relaxed plans from them. We evaluated A\*, WA\*, AWA\*, MAWA\*, AWA\*-AC, AWA\*-PD, AWA\*-AC-LE, FF, HSP-2, STAN 4, and AltAlt on several problems from many domains.

AWA\* and MAWA\* outperform STAN 4 and AltAlt on all problems. AWA\*, AWA\*-AC and MAWA\* also outperform FF on several problems. AWA\*-AC outperforms HSP-2 on several problems. There are several domains on which HSP-2 and FF are much faster than our variants. We reported on the insights about this using the notion of waiting time which captures the amount of delay in the expansion of a node due to a change in the heuristic evaluation strategy. We compared AWA\* with WA\* using multiple values of the static weight in WA\*. This shows that node-dependent weighting is superior to static weighting. We also ran the variants with many sets of values of their parameters. This evaluation shows that the number of problems solved by the variants in a given time limit does not change significantly despite a large variation in the parameters' values. So a user does not need to have a good knowledge of the domain or planner in order to select the values of the parameters. Our work shows that node-dependent weighting, conditional two-phase heuristic evaluation and lazy evaluation are very useful enhancements to heuristic search for state-space planning.

**Acknowledgement:** This work was funded by NSF grant IIS-0119630 to Amol Dattatraya Mali. This work was performed when Minh Tang was a graduate student at University of Wisconsin, Milwaukee. The authors thank Guy Schenk for conducting experimental comparison of AWA\* and WA\*. The authors thank Aparna Chougala for conducting an experimental comparison of the five variants of A\* with many sets of values of their parameters.

1. J. Hoffmann and B. Nebel, The FF Planning System: Fast Plan Generation through heuristic search, *Journal of Artificial Intelligence Research*, Vol. 14, 2001, pp. 253-302.
2. B. Bonet and H. Geffner, Planning as Heuristic Search, *Artificial Intelligence*, Vol.129(1-2), June 2001, pp. 5-33.
3. R. Nigenda, X. Nguyen, and S. Kambhampati, AltAlt: Combining the Advantages of Graphplan and Heuristic State Search, ASU Computer Science Technical Report, 2001.
4. F. Bacchus, AIPS'00 Planning Competition, *AI Magazine*, Vol.22 No.3, Fall 2001, pp. 47-56.
5. M. Do and S. Kambhampati, Sapa: A Domain-Independent Heuristic Metric Temporal Planner, *Proceedings of European Conference on Planning*, 2001, pp. 109-120
6. I. Pohl, The avoidance of (relative) catastrophe, heuristic competence, genuine dynamic weighting and computational issues in heuristic problem solving, *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1973, pp. 20-23.
7. Ira Pohl, Heuristic search viewed as path finding in a graph, *Artificial Intelligence*, 1(3), 1970, pp. 193-204.

8. Andreas L. Koll and Hermann Kaindl, A new approach to dynamic weighting, *Proceedings of European Conference on Artificial Intelligence (ECAI)*, 1992, pp. 16-17
9. J. Pearl, *Heuristics: Intelligent Search Strategies for Computer Problem Solving*, Addison-Wesley Publishing Company, 1984.
10. Minh Tang and Amol D. Mali, Variants of A\* for planning, *Proceedings of European Conference on Artificial Intelligence (ECAI)*, 2004, pp. 1093-1094.
11. Minh Tang and Amol D. Mali, Search control techniques for planning, *Proceedings of IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, Sacramento, Nov. 2003, pp. 168-175
12. P.E. Hart, N.J. Nilsson, B. Raphael, Correction to "A formal basis for the heuristic determination of the minimum path costs", *SIGART Newsletter*, 37, 1972, pp. 28-29.
13. XuanLong Nguyen, Subbarao Kambhampati and Romeo Sanchez Nigenda, Planning graph as the basis for deriving heuristics for plan synthesis by state space and CSP search, *Artificial Intelligence*, 135(1-2), 2002, pp. 73-123.
14. A. Blum and M. Furst, Fast Planning through planning graph analysis, *Artificial Intelligence*, Vol.90 (1-2), 1997, pp. 281-300.
15. D. Long and M. Fox, Stan4: A Hybrid Planning Strategy based on Sub-problem Abstraction, *AI Magazine*, Vol. 22 No.3, Fall 20001, pp. 81-85.
16. Joseph Culberson and Jonathan Schaeffer, Searching with pattern databases, *Computational Intelligence*, 14(3), 1998, pp. 318-334.
17. Richard E. Korf and Ariel Felner, Disjoint pattern database heuristics, *Artificial Intelligence*, Vol. 134, 2002, pp. 9-22
18. Amol Dattatraya Mali and Minh Tang, On the variants of A\*, Technical report, Computer science, University of Wisconsin, Milwaukee, Dec. 2003.
19. P. Hart, N. Nilsson and B. Raphael, A formal basis for the heuristic determination of minimum-cost paths, *IEEE Transactions on Systems, Science and Cybernetics*, 4(2), 1968, pp. 100-107
20. Minh Tang, and Amol Dattatraya Mali, Variants of A\* for planning, Technical report, Computer science, University of Wisconsin, Milwaukee, June 2003.
21. A. Felner, S. Kraus and R.Korf, KBFS: K-best-first search, *Annals of Mathematics and Artificial Intelligence*, Vol. 39, 2003, pp. 19-39
22. Y. Kitamura, M. Yokoo, T. Miyaji and S. Tatsumi, Multi-state commitment search, *Proceedings of the IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, 1998, pp. 431-439
23. David Furcy and Sven Koenig, Scaling up WA\* with commitment and diversity, *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2005, pp. 1521-1522.
24. Rong Zhou and Eric Hansen, Beam-stack search: Integrating backtracking with beam search, *Proceedings of International Conference on Automated Planning and Scheduling (ICAPS)*, Monterey, CA, June 2005, pp. 90-98.
25. W. Harvey and M. Ginsberg, Limited discrepancy search, *Proceedings of the*

- International Joint Conference on Artificial Intelligence (IJCAI)*, 1995, pp. 607-615
26. David Furcy and Sven Koenig, Limited discrepancy beam search, *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2005, pp. 125-131
27. Dimitris Vrakas and Ioannis Vlahavas, HYBRID ACE: Combining search directions for heuristic planning, *Computational Intelligence*, Vol. 21, Number 3, 2005, pp. 306-331
28. Alfonso Gerevini, Alessandro Saetti and Ivan Serina, Planning through stochastic local search and temporal action graphs in LPG, *Journal of Artificial Intelligence Research*, Vol. 20, 2003, pp. 239-290.
29. Dimitris Vrakas, Grigorios Tsoumakas, Nick Bassiliades and Ioannis Vlahavas, *HAPRC*: An automatically configurable planning system, *AI Communications*, Vol. 18, No. 1, 2005, pp. 1-20
30. Karen I. Trovato and Leo Dorst, Differential A\*, *IEEE Transactions on Knowledge & Data Engineering*, Vol. 14, No. 6, November/December 2002, pp. 1218-1229