

# Search Control Techniques for Planning

Minh Tang and Amol Dattatraya Mali

Electrical Engineering & Computer Science,

University of Wisconsin, Milwaukee, WI 53211

Phone: 1-414-229-6762, Fax: 1-414-229-2769, {minhtang,mali}@uwm.edu

## Abstract

*Significant advances have been made in heuristic search for classical planning in the last six years. Most of these planners use A\* style search. We report on two sound and complete domain-independent classical planners AWA\* (Adjusted Weighted A\*) and MAWA\* (Modified AWA\*) in this paper. AWA\* is the first planner to use node-dependent weighting in A\*. MAWA\* uses a two-phase heuristic evaluation. MAWA\* applies node-dependent weighting to a subset of the nodes in the fringe, after the two-phase evaluation. We report on an empirical comparison of AWA\*, MAWA\* with classical planners AltAlt, FF and STAN 4. Both AWA\* and MAWA\* outperform AltAlt and STAN 4. Both AWA\* and MAWA\* solve many problems that FF does not. The ideas in AWA\* and MAWA\* are general enough to be applicable in solving other planning problems like temporal planning and planning with resources and numerical variables.*

## 1 Introduction

Significant advances have occurred in heuristic search for classical planning in the last six years. These are clear from the success of planners FF [6], HSP-2 [3], and AltAlt [9] at the international planning competition in 2000 [1]. HSP-2 and AltAlt carry out weighted A\* (WA\*) style search. WA\* style search has been also used by planner Sapa for planning in domains containing metric time and resource quantities [4].

HSP-2, Sapa and AltAlt keep the weight in WA\* fixed throughout the search. Current planners use the same heuristic to evaluate all nodes in the fringe. Changing weights in WA\* can be advantageous. Using different weights for different nodes can provide better search control without loss of completeness and without significant loss of optimality. Node-dependent weights can be used to compensate for the limitations of the heuristic used. Node-dependent weights can use information about nodes that the heuristic does not capture. Using the same heuristic to eval-

uate all nodes in the fringe can be highly disadvantageous. 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 may not be useful. Using different heuristics to evaluate different nodes can be extremely 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.

We report on two sound and complete classical planners AWA\* and MAWA\* in this paper. AWA\* uses node-dependent weights which we refer to as dynamic weighting. Node-dependent weights provide an additional form of search control, besides actual and estimated path costs. AWA\* differs from other planners due to its use of dynamic weighting in A\* search. AWA\* uses same heuristic to evaluate all nodes in the fringe. MAWA\* is a modification of AWA\*. MAWA\* uses 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. Our empirical evaluation of AWA\* and MAWA\* on five domains shows that they both significantly outperform three planners that did very well at the 2000 international planning competition. The ideas of dynamic weighting and multi-phase heuristic evaluation of selected nodes in the fringe can also be used in solving other planning problems like temporal planning, planning with resource quantities and planning with numerical variables.

This paper makes the following contributions:

- We report on two sound and complete planners AWA\* and MAWA\*. AWA\* is the first planner to use node-dependent weighting in A\* search. MAWA\* is the first planner to use two-phase heuristic evaluation and node-dependent weighting for a selected set of nodes in the fringe.

- We report on an empirical evaluation of A\*, WA\*, AWA\*, and MAWA\* on several problems from five domains. 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. MAWA\* and AWA\* also outperform FF on several problems.

The paper is organized as follows. We review advances in heuristic search for classical planning in section 2. We describe AWA\* and MAWA\* in sections 3 and 4 respectively. We report on an empirical evaluation of A\*, WA\*, AWA\*, MAWA\* and a comparison with FF, AltAlt and STAN 4 in section 5. We discuss our theoretical work on more variants of A\* and directions for future work in section 6. We report on the conclusions in section 7.

## 2 Background

In this section, we describe planners AltAlt, FF and HSP-2. We also explain some key notions like relaxed planning graph, relaxed plan and statically mutually exclusive (mutex) actions.

AltAlt [8], HSP-2 [3], and FF [6] 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. The development of FF was inspired by HSP, the initial version of HSP-2. FF is a forward state-space local search-based planner. It 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 relaxed planning graph (RPG) is constructed assuming that the delete effect lists of actions are empty. An RPG has alternating action levels and proposition levels like the planning graph [2]. The 2000 competition version of FF 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. 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 serial leveled off 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. Details of the competition can be found in [1]. We have reported on an empirical comparison of AWA\* and MAWA\* with AltAlt and FF in section 5.

MAWA\* (section 4) uses the notion of statically mutex actions. Two actions  $o_i$  and  $o_j$  are statically 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  $p$ .

## 3 AWA\*

Planning as heuristic search has been shown to be quite effective, with exemplary planners like FF [6], HSP-2 [3], and Sapa [4]. The search algorithm employed by Sapa and HSP-2 is very similar to weighted A\* (WA\*). Weighted A\* is a variant of A\* [5]. It's a well known fact, discussed in [10], that the A\* algorithm 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. A\* may not perform well despite the use of weight. 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 making sure that the search time is still acceptable.

The A\* search algorithm 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. A\* is complete if there are finite number of nodes whose  $f()$  values are less than the cost of the optimal solution. If  $h$  is admissible, then A\* is optimal. The weighted variant of A\* uses the following equation

$$f(n) = g(n) + \omega * h(n), \quad \omega \geq 0$$

Adjusted weighted version of A\* 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 [11], 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 search progresses. Though the weighting in AWA\* and the one in [11] are dynamic, there is an important difference between these two schemes. The weighting in AWA\* is node-dependent. The weighting in [11] is not node-dependent.

A version of A\* using dynamic weighting for  $h()$  is mentioned on page 88 in [10]. 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\*, 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 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.

AWA\* is complete. The proof of this is given in our longer technical report [12]. If the heuristic  $h$  is admissible, the cost  $C$  of the solution found by AWA\* satisfies

$$C \leq \max \left\{ 1, \frac{\mu_h}{\mu_g} \right\} C^* \quad (1)$$

where  $C^*$  is the cost of the optimal solution. The proof of this is given next.

**Proof:** The maximum value of  $\psi(n) = \mu_h - \frac{s(n)}{\delta_h |G|}$  occurs when  $s(n) = 0$ , so that  $\psi(n) = \mu_h$ . Therefore, the  $f$  value in AWA\* satisfies

$$f(n) \leq \mu_g * g(n) + \mu_h * h(n)$$

Now let  $f'(n) = \mu_g * g(n) + \mu_h * h(n)$ .  $f'(n)$  can then be rewritten as  $f'(n) = g(n) + (\mu_h / \mu_g) * h(n)$ . This is because the cost of solution found with  $f'(n) = \mu_g * g(n) + \mu_h * h(n)$  is same as the cost of solution found with  $f'(n) = g(n) + \frac{\mu_h}{\mu_g} * h(n)$ . So if  $C'$  is the cost of the solution found by using  $f'(n) = g(n) + (\mu_h / \mu_g) * h(n)$  and  $C^*$  is the cost of the optimal solution, then  $C' \leq (\mu_h / \mu_g) * C^*$ . Since

$f(n) \leq f'(n)$  for all  $n$ , the cost  $C$  of the solution found by using  $f(n)$  in AWA\* must satisfy

$$C \leq C' \leq \frac{\mu_h}{\mu_g} * C^*$$

Since  $(\mu_h / \mu_g)$  might be less than 1, we have the specified upper bound on the cost.

## 4 MAWA\*

A factor that might adversely affect the performance of AWA\* is that weights are computed for all nodes. Another potential problem in AWA\* is that if every node is considered for weight adjustment, then some nodes that shouldn't have the weight of their  $h()$  adjusted also have it adjusted. Therefore, we propose another method of dynamic weighting which uses weights for only more promising nodes. The set of these more promising nodes is the *candidate set*. This approach helps to reduce the computational time associated with the weights of nodes.

Yet another 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 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 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 two-phase heuristic evaluation and node-dependent weights for only selected nodes. MAWA\* algorithm is described below.

- 1(a). **(Phase 1)** For every node  $n$  in the fringe, 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  $h'(n)$  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.

In our MAWA\* implementation,  $h(n)$  is same as the number of actions in the relaxed plan at  $n$ , as in FF [6]. 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 statically mutex actions in  $i$  th step of the relaxed plan at  $n$ . This adjustment can give us some more information. A node with a high number of static action 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 notion of candidate set  $S$  is mentioned in [10]. Pearl 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.

MAWA\* is complete. The proof of completeness of MAWA\* is similar to the proof of completeness of AWA\*. MAWA\* is inspired by the  $A_\epsilon^*$  scheme described in [10]. Let  $C^*$  be the cost of optimal solution found by A\*. The cost  $C$  of the solution found by MAWA\* satisfies

$$C \leq (1 + \epsilon) * C^*$$

if  $h$  and  $h'$  in step 1 of MAWA\* are admissible. The proof of this is given next. The proof is a modification of the proof from page 89 in [Pearl 1984]. In the proof,  $X = \{x_1, x_2, x_3 \dots, x_n\}$  is the set of all unexpanded nodes in the search space. We denote the minimum  $f$  value among all  $x_i \in X$  by  $f_{min}$ , i.e.  $f_{min} \leq f(x_i) \forall x_i \in X$ . We define the candidate set  $S \subseteq X$  as follows

$$S = \{x_i | f(x_i) \leq (1 + \epsilon) * f_{min}\}$$

**Proof:** We assume that  $h$  and  $h'$  are admissible. Let  $x_0 \in X$  be the node such that  $f(x_0) = f_{min}$ . Also let  $x_1 \in X$  be the node that lies on the optimal solution path. Let  $t \in S$  be the node chosen for expansion that was found to contain the goal. Since  $x_1$  lies on the optimal solution path and  $h$  and  $h'$  are both admissible,  $f(x_1) \leq C^*$ . We must also have that  $f(x_0) = f_{min} \leq f(x_1)$ . Since  $t \in S$ ,  $f(t) \leq (1 + \epsilon) * f_{min} = (1 + \epsilon) * f(x_0)$ . Using all of the above information together, we have

$$C = f(t) \leq (1 + \epsilon) * f(x_0) \leq (1 + \epsilon) * f(x_1) \leq (1 + \epsilon) * C^*$$

What the above result means is that the optimality of MAWA\* can be even better than that of AWA\*. That's, MAWA\* is able to incorporate the advantages of AWA\* and can also improve the optimality and the computational cost. If  $h$  and  $h'$  are admissible, the cost of solution found by MAWA\* is bounded above by  $(1 + \epsilon) * C^*$ . The cost of

implemented in C++ and the version of AltAlt that we used is also implemented in C++. All planners were run on the same machine mentioned at the start of this section. FF versions 1.0 and 2.3, AltAlt, A\*, WA\*, AWA\* and MAWA\* all find serial plans. STAN 4 finds parallel plans.

### 5.1 Domain Descriptions

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

- **Blocks-world:** Blocks-world using the  $move(x,y,z)$  operator where  $x \neq y, y \neq z, x \neq z, x \neq Table$ . The operator moves block  $x$  from  $y$  to  $z$ . With  $n$  blocks, there are  $n \cdot (n - 1) \cdot (n - 2) + 2 \cdot n \cdot (n - 1)$  ground instances of this operator.
- **Blocks-inversion:** A variant of the Blocks-world domain using the  $move(x,y,z)$  operator. 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 domain 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-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 she/he has enough cash. Cash can be found only at a small number of cities. This domain too contains non-invertible actions.

The problems from the Blocks-world and Block-inversion domains were manually generated. Problems in the TSP-1, TSP-2 and TSP-3 domains were automatically generated. All problems have solutions.

### 5.2 A\*, WA\*, and AWA\*

Table 1 reports the performance of A\*, WA\* and AWA\* on 8 problems from the Blocks-world domain. 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. P# in the first column in tables 1, 2 and 6 denotes problem number. acts in tables 1,2,4 and 5 denotes the number of actions in the plans found.

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. Each connection allows travel in both directions. The actual connections are randomly generated. 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 fastest on 7 out of 8 problems.

P#	A*		WA*		AWA*	
	acts	time	acts	time	acts	time
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 1. A\*, WA\*, AWA\* on Blocks-world

P #	A*		WA*		AWA*	
	acts	time	acts	time	acts	time
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 2. A\*, WA\*, AWA\* on TSP-1

### 5.3 AWA\*, FF, and STAN 4

Table 3 reports results of AWA\*, FF version 1.0, and STAN 4 on 9 problems from the Blocks-inversion domain.  $a$  in tables 3 and 6 denotes the number of actions in the plans found.  $\# B$  in table 3 denotes the number of blocks in the problems.  $S$  in the last column of table 3 and tables 4 and 5 denotes speedup.  $t$  in tables 3 and 6 denotes the solving times. We also report the speedup of AWA\* with respect to FF. The problems in Table 3 had between 10 and 18 blocks. 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. All problems solved by FF and STAN 4 were solved by all 12 instantiations of AWA\*. All 12 instantiations of AWA\* solved more problems than FF version 1.0 and STAN 4. AWA\* was more effective than FF and STAN 4 for a wide range of values of  $\delta_h$  and  $\mu_h$ .

# B	AWA*		FF		STAN 4		S
	a	t	a	t	a	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

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

### 5.4 MAWA\*

Table 4 reports results of AWA\* and MAWA\*. The first 8 problems are from the Blocks-world domain. The next 4 problems are from the TSP-2 domain. The last 4 problems are from the TSP-3 domain. The problems from the Blocks-world domain 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 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 of MAWA\* with respect to 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.

The results on the 61 problems in the five tables show that AWA\* and MAWA\* are both able to easily find plans containing around 45 actions. Both planners can easily find larger plans on problems easier than the ones we used. Since MAWA\* computes weights for a smaller number of nodes than AWA\*, the running time of MAWA\* is expected to be lower than that of AWA\* on most problems. Table 4 shows that this is indeed the case for 15 out of 16 problems. It is hard to empirically compare optimality of A\*, WA\* and AWA\*, since AWA\* solved many more problems than A\* and WA\*. AWA\* and MAWA\* found shorter plans than FF on many problems.

### 5.5 Additional Results

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 the Blocks-world domain, and the remaining 8 problems are from the TSP-1 domain. The Blocks-world domain 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

Prob #	AWA*		MAWA*		S
	acts	time	acts	time	
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 4. AWA\*, MAWA\* on Blocks-world, TSP-2 and TSP-3**

Prob #	MAWA*		FF		S
	acts	time	acts	time	
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+	-

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

210 connections. The actual connections 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.

P#	AWA*		FF		STAN 4	
	a	t	a	t	a	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+

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

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 [12].

## 6 Discussion

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

Sapa [4] uses weighted A\* for planning. It is very effective at generating high quality plans in domains containing durative actions and resource quantities. Our two-phase heuristic evaluation can be viewed as a special case of 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. Our ideas of varying  $\epsilon, c, \omega$ , during search and using  $n$ -phase heuristic evaluation,  $n > 2$  and varying  $n$  during search, can be used in solving other planning problems like temporal planning and planning with resource quantities. In [13], we report on 9 variants of  $A^*$ . One of them is same as  $AWA^*$ . One variant is a variant of  $MAWA^*$  which uses  $n$ -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. 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 bounds on the costs of solutions found by the nine variants of  $A^*$  in [13], assuming that the heuristics are inadmissible. Empirically evaluating these algorithms on planning problems is our future work.

## 7 Conclusion

We reported on two variants of the  $A^*$  algorithm for planning. Both variants ( $AWA^*$  and  $MAWA^*$ ) are sound and complete.  $AWA^*$  uses node-dependent weights to adjust the heuristic estimates.  $MAWA^*$  uses a two-phase heuristic evaluation.  $MAWA^*$  uses a computationally cheap heuristic in phase 1 and it adjusts the heuristic estimates of some nodes in phase 2, using a different heuristic. The 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. We evaluated  $A^*$ ,  $WA^*$ ,  $AWA^*$ ,  $MAWA^*$ , FF, STAN 4, and AltAlt on several problems from five domains.  $AWA^*$  and  $MAWA^*$  outperform STAN 4 and AltAlt on all problems.  $AWA^*$  and  $MAWA^*$  also outperform FF on several problems. Given that variants of  $A^*$  are very effective in generating high quality plans in temporal planning with resource quantities, variants of  $AWA^*$  and  $MAWA^*$  for such more expressive problems are worthy of investigation. Such variants of  $AWA^*$  and  $MAWA^*$  can be developed by varying  $\epsilon$  and/or using node-dependent weights and/or varying  $n$  in  $n$ -phase heuristic evaluation and/or varying  $c$  and/or varying the size of the candidate set.

**Acknowledgement:** This work is funded by NSF grant IIS-0119630 (Pruning Techniques for More Expressive Planning) to Amol Mali.

[1] F. Bacchus, AIPS'00 Planning Competition, *AI Magazine*, Vol.22 No.3, Fall 2001, pp. 47-56.

[2] A. Blum and M. Furst, Fast Planning through planning graph analysis, *Artificial Intelligence*, Vol.90 (1-2), 1997,

pp. 281-300.

[3] B. Bonet and H. Geffner, Planning as Heuristic Search, *AI Journal*, Vol.129(1-2), June 2001, pp. 5-33.

[4] M. Do and S. Kambhampati, Sapa: A Domain-Independent Heuristic Metric Temporal Planner, *Proceedings of European Conference on Planning*, 2001, pp. 109-120

[5] 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.

[6] J. Hoffmann, FF: The Fast Forward Planning System, *AI Magazine*, Vol.22 No.3, Fall 2001, pp. 57-62.

[7] 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.

[8] X. Nguyen and S. Kambhampati, Extracting Effective and Admissible State Space Heuristics from the Planning Graph, *AAAI proceedings*, 2000, pp. 798-805.

[9] R. Nigenda, X. Nguyen, and S. Kambhampati, AltAlt: Combining the Advantages of Graphplan and Heuristic State Search, ASU Computer Science Technical Report, 2001.

[10] J. Pearl, *Heuristics: Intelligent Search Strategies for Computer Problem Solving*, Addison-Wesley Publishing Company, 1984.

[11] I. Pohl, The avoidance of (relative) catastrophe, heuristic competence, genuine dynamic weighting and computational issues in heuristic problem solving, *IJCAI proceedings*, 1973, pp. 20-23.

[12] Minh Tang, and Amol Dattatraya Mali, Variants of  $A^*$  for planning, Technical report, Computer science, University of Wisconsin, Milwaukee, June 2003.

[13] Amol Dattatraya Mali and Minh Tang, On the variants of  $A^*$ , Submitted to the *Journal of ACM* in August 2003.