# AWA* - A Window Constrained Anytime Heuristic Search Algorithm

**Sandip Aine    P. P. Chakrabarti    Rajeev Kumar**
Department of Computer Science & Engineering
Indian Institute of Technology Kharagpur, India.
{sandip, ppchak, rkumar}@cse.iitkgp.ernet.in

## Abstract

This work presents an iterative anytime heuristic search algorithm called Anytime Window A* (AWA*) where node expansion is localized within a sliding window comprising of levels of the search tree/graph. The search starts in depth-first mode and gradually proceeds towards A* by incrementing the window size. An analysis on a uniform tree model provides some very useful properties of this algorithm. A modification of AWA* is presented to guarantee bounded optimal solutions at each iteration. Experimental results on the 0/1 Knapsack problem and TSP demonstrate the efficacy of the proposed techniques over some existing anytime search methods.

## 1 Introduction

Development of optimization methods that can work within time limitations is a major need for modern practitioners facing large-sized problems that are NP-hard in nature. This requirement has spawned research in the direction of designing anytime algorithms [Dean and Boddy, 1988], i.e., the algorithms which can work under any *time* limitations. Anytime algorithms are expected to regularly produce solutions of improved quality and when suddenly terminated, return the best solution produced so far as the result.

Among heuristic search techniques, the depth-first branch and bound (DFBB) approach and its variants [Sarkar *et al.*, 1991; Zhang, 1998] are by default anytime algorithms, as they produce a stream of gradually improving solutions. However, in general, the depth-first techniques suffer from excessive node expansions before they actually reach the optimal solution.The best-first anytime search algorithms can be broadly classified into two categories, namely, weighted heuristic search [Pohl, 1970; Hansen *et al.*, 1997; Zhou and Hansen, 2002; Likhachev *et al.*, 2004] and beam search [Zhou and Hansen, 2005; Furcy, 2006].

In weighted heuristic search, the heuristic estimate is multiplied by a weight, $w$ ($w \geq 1$), to yield $f(n) = g(n) + w * h(n)$. In the anytime variants of weighted A*, a series of solutions are produced either by iterating with the same chosen weight [Hansen *et al.*, 1997; Zhou and Hansen, 2002] or by gradually decrementing the weight using a chosen $\Delta$ after each iteration [Likhachev *et al.*, 2004]. Another algorithm was proposed to modify the weighted A* technique with selective commitments [Kitamura *et al.*, 1998]. One of the major disadvantages of using the weighted A* based techniques is that weighing the heuristic introduces non-admissibility. As the basic characteristics of A* change substantially with non-admissible heuristics [Pearl, 1984], it becomes very difficult to establish a relation between node expansions and the weighing factors ($w$,$\Delta$). Therefore, to use a weighted A* technique, lengthy simulations are required to find appropriate parameter settings. Also, there are applications where no specific heuristic for the remaining path is available, the search may be guided only by the $g(n)$ information [Pearl, 1984] or a lower bound $f(n)$-value may be used without distinct $g(n)$ and $h(n)$ components [Kumar *et al.*, 1995]. In such cases, the weighted A* techniques are not directly applicable.

On the other hand, in beam search, a chosen number of most promising nodes at each level of the search tree/graph are selected for further expansion, and the remaining nodes are pruned. Several anytime variants of the basic beam search technique [Zhang, 1998; Zhou and Hansen, 2005; Furcy, 2006] have been applied for different problems with reasonably good results. However, the lack of models, which can provide the user an apriori estimate about the quality-time trade-off, remains a problem for these algorithms as well.

In this work, we present an alternative anytime heuristic search algorithm Anytime Window A* (AWA*), which localizes the global competition performed by A* within a fixed-size window comprising of levels of the search tree/graph. The nodes which lie outside the active window are not considered for expansion at that point. The window is slid downwards to provide a depth-first component to the best-first search that occurs within a window. The window size is set to 0 to start with (depth-first mode) and is then increased iteratively proceeding towards pure best-first search algorithms like A*. The solution quality is expected to improve for iterations with a larger window size. While in AWA*, we use the basic anytime principles of depth guided pruning and iterative relaxation, the algorithm retains the admissibility of the heuristic estimate while it continues to search in best-first mode within the given window. It may be noted that AWA* does not need explicit decomposition of the evaluation function ($f(n)$) into $g$ and $h$ components. We analyze the characteristics of the presented algorithm using a uniform search tree model. This reveals some very interesting properties of the algorithm in terms of heuristic accuracy versus solution quality and related search complexity.

We also present a bounded optimal version of the AWA* algorithm, called Bounded Quality Anytime Window A* (BQAWA*), which is guaranteed to produce solutions within a pre-specified sub-optimality bound, at each iteration. In BQAWA*, the search backtracks and dynamically adjusts the window size within an iteration, such that the bound restriction is adhered to.

We demonstrate the efficacy of the presented strategies on two optimization problems, namely, the traveling salesman problem (TSP) and the 0/1 knapsack problem. Comparative results with the existing anytime techniques (like DFBB, ARA*, Beam-stack search) show considerable improvement in performance.

## 2 Anytime Window A* Algorithm

The algorithm A* considers each node to be equivalent in terms of information content and performs a global competition among all the partially explored paths to select a new node. In practice, the heuristic errors are usually distance dependent [Pearl, 1984]. Therefore, the nodes lying in the same locality are expected to have comparable errors, where as the error may vary substantially for nodes which are distant to each other. Thus, if the global competition performed by A* is localized within some boundaries, tighter pruning can be obtained. Also, as *better* nodes at a level are generally very competitive even within small localities, we can expect good quality solutions with localized expansions. This observation forms the basis of AWA*, where we localize the node expansions using a sliding window. The AWA* algorithm works in two phases, in the inner loop it uses the Window A* (Algo. 1) routine with a fixed window size and computes the solution under the current restriction. In the outer loop (Algo. 2) the window restrictions are gradually relaxed to obtain iteratively improving solutions.

The Window A* routine works with a pre-specified window size. This window size is used to restrict the active queue of the A* algorithm. For a given window size $\omega$, the expansions are localized in the following way. When the first node of any level (say $l$) is expanded, all nodes in the open list which are from level $(l - \omega)$ or less will be suspended for this iteration. The window slides in a depth-first manner when deeper nodes are explored. Window A* terminates when the first goal node is expanded (A* like termination) or if the open list does not contain any node having estimated cost less than the cost of the best solution obtained in previous iterations.

The AWA* algorithm calls the Window A* routine multiple times with gradual relaxation of the window bounds. At the start, window size is set to 0 (depth-first mode). Once the Window A* routine terminates, AWA* checks whether there are any nodes suspended in the previous iteration. If the suspended list is empty, the algorithm is terminated returning the optimal solution. Otherwise, the open list is emptied and the suspended list is moved to the open list. The window size is increased in a pre-decided manner (we have used gradual increment by 1), and Window A* is called again. If the algorithm is terminated through external interrupts, the best solution obtained thus far is returned.

**Algorithm 1 Procedure Window A***

```
1: $CurLevel \leftarrow -1$;
2: if $OpenList$ is empty then
3:    Return $BestSol$;
4: end if
5: Select node $n$ which has the least $f(n)$-value node from $OpenList$;
6: Insert $n$ into $ClosedList$;
7: if $f(n) \geq BestSol$ then
8:    Return $BestSol$;
9: else if $Level(n) \leq CurLevel - WindowSize$ then
10:    Remove $n$ from $ClosedList$; Insert $n$ into $SuspendList$;
11:    goto line 2
12: else if $Level(n) > CurLevel$ then
13:    $CurLevel \leftarrow Level(n)$
14: end if
15: if IsGoal($n$) then
16:    $BestSol \leftarrow f(n)$; $Goal \leftarrow n$;
17:    Return $BestSol$;
18: end if
19: for Each successor node $n'$ of $n$ do
20:    Calculate $f(n')$;
21:    if $n'$ is not in $OpenList$ or $ClosedList$ or $SuspendList$ then
22:       $Parent(n') \leftarrow n$; $Level(n') \leftarrow Level(n) + 1$;
23:       Insert $n'$ to $OpenList$;
24:    else if $n'$ is in $OpenList$ or $SuspendList$ then
25:       if $f(n') <$ previously calculated path estimation then
26:          $Parent(n') \leftarrow n$; Update $f(n')$, $Level(n')$;
27:       end if
28:    else if $n'$ is in $ClosedList$ then
29:       if $f(n') <$ previously calculated path estimation then
30:          $Parent(n') \leftarrow n$; Update $f(n')$, $Level(n')$;
31:          Insert $n'$ to $OpenList$;
32:       end if
33:    end if
34: end for
35: goto line 2
```

**Algorithm 2 Anytime Window A* (AWA*)**

```
1: INPUT :: A search tree/graph and a start node $s$.
2: $ClosedList \leftarrow \phi$; $SuspendList \leftarrow \phi$; $WindowSize \leftarrow 0$; Calculate
   $f(s)$; $Level(s) \leftarrow 0$; Insert $s$ to $OpenList$; $BestSol \leftarrow \infty$;
3: $BestSol \leftarrow$ Window A*(); {Call Window A*}
4: if $SuspendList$ is empty then
5:    Terminate with $BestSol$ as the Optimal Solution;
6: else
7:    Add $OpenList$ to $ClosedList$; Move $SuspendList$ to $OpenList$;
      $WindowSize \leftarrow WindowSize + 1$;
8:    goto line 3
9: end if
```

For an anytime algorithm, there are two desirable properties, namely, *interruptibility* and *progress*. The first property ensures the anytime nature, i.e., the algorithm should provide a solution at pre-mature terminations. The second one states that the solution quality should improve with time, and if enough time is allocated the algorithm should eventually return the optimal solution. For finite search spaces, AWA* adheres to both the properties mentioned. It starts in a depth-first mode (with no back-tracking) trying to obtain fast (possibly sub-optimal) solutions, and then iteratively increases the window size to produce better quality solutions. Eventually, when the window size becomes such, that all the 'optimal-path' nodes lie within the active window, the optimal solution is returned.

While iterative algorithms provide opportunities to attain different trade-offs through anytime terminations, the search efforts can increase substantially if re-expansions are not controlled. For AWA* with a consistent heuristic, a node is expanded at most once within the Window A* routine, and it can only be re-expanded in a later iteration if its $g$-value is lowered. Thus AWA* requires minimal re-expansion.

It may be noted, that in the generalized case AWA* does not guarantee a solution at each iteration (Window A* loop). If this property is a requirement; an easy way to attain this is through back-tracking (i.e., by moving the suspended list to open list at line 2 of Algo. 1), when no solution is found. However, we propose a better method for this in BQAWA* (Sec. 4), which not only provides a solution but guarantees a bounded quality solution, at each iteration.

## 3 Analyzing AWA* using a Search Tree Model

In this section, we analyze the quality-time response of Window A* algorithm in terms of a uniform cost tree model (as suggested in [Pearl, 1984]). We model the search space as a uniform $m$-ary tree $T$, with a unique start state $s$. The leaf nodes are grouped into two parts. There is a unique optimal goal node $g$, situated at a distance $N$ from $s$, and all other nodes at level $N$ are sub-optimal terminal nodes. Window A* terminates when either the optimal goal node or a sub-optimal terminal node is reached. The optimal solution path is given as $(s, n_{g,1}..n_{g,i}..n_{g,N-1}, g)$ where $n_{g,i}$ denotes the solution path node at level $i$. The trees $T_1 ... T_i ... T_N$ are 'off-course' sub-trees of $T$, one level removed from the solution path. An 'off-course' node is labeled as $n_{i,j}$, where $j$ denotes the level of that node and $i$ denotes the root level of the 'off-course' sub-tree ($T_i$) to which the node belongs. Using this

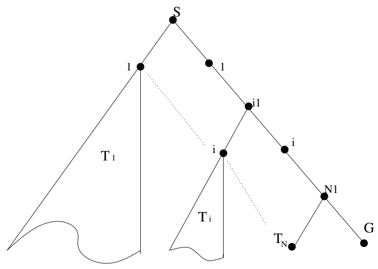

Figure 1: Uniform binary tree (model)

model (Fig. 1), we try to quantify the expected node expansions and the probability of reaching the optimal goal node, for a specific window restriction. First we present few terminologies,

**Definition 1.** $q_{i,j,\omega}$ : *$q_{i,j,\omega}$ denotes the expansion probability of an 'off-course' node $n_{i,j}$ when it is in the open list and the window size is $\omega$,*

$$q_{i,j,\omega} = P(f(n_{i,j}) \leq Min(\Gamma)) \text{ , } \Gamma = min(l + \omega, N)$$
$$\textit{and, } Min(l) = min(f(n)), \forall n \textit{ such that } n \in \textit{ open list and } level(n) = l. \tag{1}$$

**Definition 2.** $r_{g,j,\omega}$ : *$r_{g,j,\omega}$ denotes the expansion probability of a 'on-path' node $n_{g,j}$ when it is already in the open list and the window size is $\omega$,*

$$r_{g,j,\omega} = P(f(n_{g,j}) \leq Min(\Gamma)), \Gamma = min(l + \omega, N) \tag{2}$$

Next, we use the search tree model to obtain the expressions for the expected node expansions and convergence probability for a given $\omega$. The expected node expansions and convergence probabilities (for a given window size) are presented in the following lemmas,

**Lemma 1.** *Expected number of node expansions $EN(\omega)$ (window size = $\omega$) for a $m$-ary uniform cost search tree $T$ (with depth $N$) is given as,*

$$EN(\omega) = \textstyle\sum_{l=0}^{l=N} P_g(g, l, \omega) + (m-1)\Sigma_{i=1}^{i=l} m^{l-i} * P_n(i, l, \omega)$$
$$\textit{Where, } P_g(g, l, \omega) = \Pi_{j=0}^{j=l} r_{g,j,\omega}, \textit{ and,}$$
$$P_n(i, l, \omega) = \Pi_{j=i}^{j=l} q_{i,j,\omega} * \Pi_{k=0}^{k=i-1} r_{g,k,\omega} \tag{3}$$

**Lemma 2.** *Expected probability of reaching the optimal goal node $P_s(\omega)$ (window size = $\omega$) for a $m$-ary uniform cost search tree $T$ (with depth $N$) is given as,*

$$P_s(\omega) = \Pi_{j=0}^{j=N} r_{g,j,\omega} \tag{4}$$

We try to estimate the expansion probability values $(q_{i,j,\omega}, r_{g,j,\omega})$ from the heuristic information available. We assume that the relative estimation errors (Eqn. 5), are independent random variables $Y(n)$ within $(0, e)$ bound $(0 \leq e \leq 1)$ with an arbitrary distribution function.

$$Y(n) = [h^*(n) - h(n)]/h^*(n) \tag{5}$$

Now, for a node $n_{i,j}$ at depth $j$ and belonging to 'off-course' sub-tree $T_i$ (Figure. 1), we have

$$g(n_{i,j}) = j,$$
$$h^*(n_{i,j}) = N + j + 2(1 - i) \tag{6}$$

Similarly, for an 'on-path' node $n_{g,j}$, we have

$$g(n_{g,j}) = j,$$
$$h^*(n_{g,j}) = N - j \tag{7}$$

Thus, with the chosen error model (Eqn. 5),

$$f(n_{i,j}) = N + 2(j - i + 1) - Y(n)(N + j + 2(1 - i))$$
$$f(n_{g,j}) = N - Y(g)(N - j) \tag{8}$$

Where $Y(n)$, $Y(g)$ are identically distributed random variables within $(0, e)$ bounds. Combining Eqn. 8 with the earlier definitions, we obtain

$$q_{i,j,\omega} = 1 - P(Y(n) \leq (j + \alpha - Min(\Gamma))/\alpha)$$
$$\text{where, } \alpha = N + j + 2(1 - i) \text{ and, } \Gamma = min(l + \omega, N) \tag{9}$$

and,

$$r_{g,j,\omega} = 1 - P(Y(n) \leq (N - Min(\Gamma))/(N - j)) \tag{10}$$

From Eqns. 1 and 2, we observe that apart from the $f(n)$ value, the node expansion probability also depends on the $Min(l + \omega)$ value. Now, the $Min(l)$ value at a given level is bounded by the minimum and maximum possible $f$-values for that level. Also, for a consistent heuristic the $Min(l)$

function should be non-decreasing with $l$. Thus, we obtain the following properties for $Min(l)$.

$$\begin{array}{l} N - e(N-l) \leq Min(l) \leq 2l + N \\ Min(l) \geq Min(l-1) \end{array} \tag{11}$$

With these observations, we represent the $Min(l)$ function as follows,

$$\begin{array}{l} Min(l) = max(Min(l-1), \\ N(1 - e(1 - s(l)) + l(e + (2-e)s(l)))) \\ \text{where, } s(l) \text{ is a function of } l, 0 \leq s(l) \leq 1. \end{array} \tag{12}$$

The $Min(l)$ function denotes the minimum value of a set of numbers. From statistics, we know that with the increase of the set cardinality the probability of the minimum converging to the lower limit increases exponentially. The implication suggests that with increase in $l$, the probability of $Min(l)$ converging to the lower limit increases exponentially. Thus, $s(l)$ can be approximated using a decreasing function with $l$, and most probably the decrease will be exponential.

Using the above presented results, we can estimate the expected node expansion and the optimal convergence probabilities for a chosen error distribution and $s(l)$ function. We assume a simple linear distribution for $Y(n)$ and approximate $s(l)$ as an exponential function of $l$, i.e., we assume,

$$P(Y(n) \leq x) = \begin{cases} 0 \text{ if } (x < 0) \\ x/e \text{ if } (0 \leq x \leq e) \\ 1 \text{ otherwise} \end{cases} \tag{13}$$

and,

$$s(l) = c^l, \;\; 0 \leq c \leq 1 \tag{14}$$

Substituting the variables in Eqns. 3 and 4, we obtain the

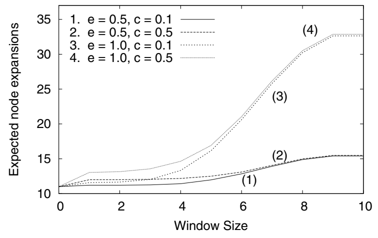

Figure 2: Expected node expansions versus window size, for a uniform binary tree of height 10 for AWA*

expected values for node expansions as well as convergence probabilities for different window constraints. In Figures 2 and 3, we present the curves for expected node expansions and convergence probabilities with different window size, for a uniform binary tree of height 10. The expected values are computed using two error boundaries ($e = 0.5$ and $e = 1.0$), and two chosen constants ($c = 0.1$ and $c = 0.5$). From the curves, we observe that the node expansions and convergence probabilities obtained are almost independent of the choice of $c$. This can be explained by the exponential nature of chosen $s(l)$ function, which rapidly converges to the lower limit. Investigating the trends with different heuristic errors, we observe that the convergence probability versus window

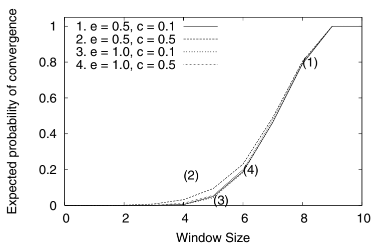

Figure 3: Convergence probability versus window size, for a uniform binary tree of height 10 for AWA*

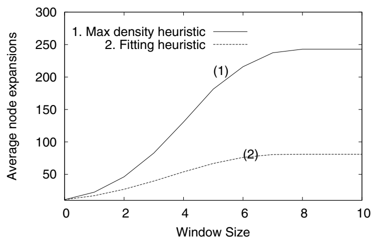

Figure 4: Average node expansions versus window size, 0/1 Knapsack problem

size curves (Fig. 3) do not show much variability with different errors. While the convergence probability for a given window size increases with heuristic accuracy, the improvement is marginal. However, the expected node expansions for a given window increases substantially with more error (Fig. 2). This observation can be explained by the fact that with more error, both the $Min(l)$ and the $f(n)$ values are expected to decrease, causing relatively small changes in the $q_{i,j,\omega}$ and $r_{g,j,\omega}$ probabilities. The difference is even less pronounced for $r_{g,j,\omega}$ as the $f(n)$ values of the 'optimal-path' nodes are expected to remain close to the $Min(l)$ values. On the other hand, the expected node expansions increase exponentially with the increase in $q_{i,j,\omega}$ values (Eqn. 3). Thus, with increased error more 'off-course' nodes are expanded, causing appreciable variance in the node expansions.

We conclude this section by presenting the experimental results for average node expansions and convergence probabilities with changing window size, for the 0/1 Knapsack problem using two different heuristics $h1$ (max. density) and $h2$ (fitting based), such that $h2$ dominates $h1$ (Fig. 4 and 5).

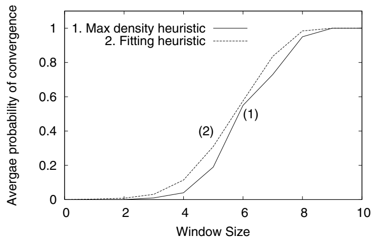

Figure 5: Average optimal convergence versus window size, 0/1 Knapsack problem

The figures depict that the average node expansions and the convergence probabilities with different heuristics show identical trends to that obtained from the analytical model.

## 4 BQAWA* - Anytime Window A* with Provable Quality Bounds

In AWA*, we have an anytime algorithm which attempts to produce a stream of gradually improving solutions. However, no bound on the obtained solution is established. In this section, we present a modified version of the AWA*, termed as Bounded Quality Anytime Window A* (BQAWA*), where each solution produced will be bounded within a pre-specified factor ($\epsilon$) of the optimal solution, the bound is iteratively tightened to produce an iteratively improving stream of solutions.

**Algorithm 3 Procedure Bounded Quality Window A***

```
1: $CurLevel \leftarrow -1$; $MinSus \leftarrow \infty$;
2: if $OpenList$ is empty then
3:     goto line21
4: end if
5: Select node $n$ which has the least $f(n)$-value node from $OpenList$;
6: Insert $n$ into $ClosedList$;
7: if $f(n) \geq BestSol$ or $f(n) \geq MinSus * \epsilon$ then
8:     goto line 18 {Update window size and back-track}
9: else if $Level(n) \leq CurLevel - WindowSize$ then
10:    Move $n$ from $ClosedList$ to $SuspendList$; Update $MinSus$;
11:    goto line 2
12: else if $Level(n) > CurLevel$ then
13:    $CurLevel \leftarrow Level(n)$
14: end if
15: GOAL EVALUATION {Same as Window A*}
16: EXPANSION {Same as Window A*.}
17: goto line 2
18: if $f(n) < BestSol$ then
19:    Move $n$ from $ClosedList$ to $OpenList$;
20: end if
21: Merge $SuspendList$ with $OpenList$; Increment $WindowSize$;
22: if $OpenList$ is empty then
23:    Return $BestSol$;
24: end if
25: goto line 1
```

To achieve the pre-specified bounds at each iteration, we modify the Window A* routine to obtain Bounded Quality Window A* or BQWA* (Algo. 3). In BQWA* routine, at any intermediate point, the minimum $f$-value among the suspended nodes ($min(f_{sus})$) is noted. If the $f$-value of the top node in the open list is greater than $\epsilon * min(f_{sus})$ (for minimization problems), the open list and the suspended list are merged, and the search back-tracks after incrementing the window size. The first solution produced by the algorithm is published with $\epsilon$ bound guarantee. The BQAWA* algorithm

**Algorithm 4 Bounded Quality Anytime Window A***

```
1: INPUT :: A search tree/graph, a start node $s$ and an initial bound $\epsilon$.
2: $ClosedList \leftarrow \phi$; $SuspendList \leftarrow \phi$; $WindowSize \leftarrow 0$; Calculate
   $f(s)$; $Level(s) \leftarrow 0$; Insert $s$ to $OpenList$; $BestSol \leftarrow \infty$;
3: $BestSol \leftarrow$ Bounded Quality Window A*();
4: if $SuspendList$ is empty then
5:     Terminate with $BestSol$ as the Optimal Solution;
6: else
7:     Add $OpenList$ to $ClosedList$; Move $SuspendList$ to $OpenList$;
8:     $\epsilon \leftarrow \epsilon - \Delta$;
9:     goto line 3
10: end if
```

(Algo. 4) works with the same principle of ARA* technique, i.e., the algorithm starts with a high initial bound ($\epsilon$) and iteratively reduces the bound to get improved solutions. It uses the BQWA* routine (Algo. 3) to obtain intermediate solutions within the chosen sub-optimality bound. After each iteration, the suspended list is moved to the open list, weight bound is reduced and BQWA* is re-run to get improved solutions. The algorithm terminates if the suspended list is empty after an iteration.

## 5 Experimental Results

In this section, we present empirical results comparing the performance of AWA* with some existing anytime techniques (DFBB, ARA* and Beam-stack search). We performed experiments with two optimization problems, namely 0/1 Knapsack and Euclidean Traveling Salesman (TSP) problem. Since time spent is usually proportional with the number of nodes expanded, we present the results in terms of node expansions.

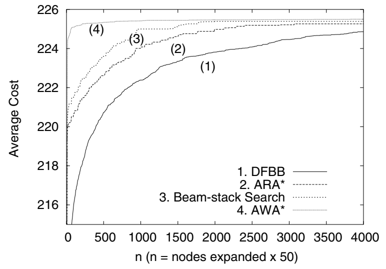

Figure 6: Average cost versus node expansions - 0/1 Knapsack problem

For 0/1 Knapsack, we estimated the heuristic by fitting the remaining objects in decreasing order of their cost-densities. We performed experiments on a set of 100 random instances of the 50-object $0/1$ Knapsack problem. Weight and costs of individual objects were generated randomly, while the constraint was chosen within $0.4 - 0.6$ of the sum of weights. In Figure 6, we present the cost versus nodes expansion results obtained for DFBB, ARA*, Beam-stack search and AWA*. Figure 7 includes the percentage optimal convergence versus node expansion curves. For ARA*, the weight limits were initialized to 2.0, and decreased by 0.1 per iteration.

For Euclidean TSP, we used the minimum spanning tree (MST) heuristic. We performed the experiments on a randomly generated set of 100 25-city problem instances. The intermediate tour length and percentage optimality results comparing AWA* with DFBB, ARA* and Beam-stack search are included in Figures 8 and 9, respectively.

From the results obtained for both 0/1 Knapsack and TSP, we observe that the performance of AWA*, is superior to all, DFBB, ARA* and Beam-stack search, both in terms of intermediate solution qualities as well as percentage optimal convergences.

In Table 1, we present the average node expansions for ARA* and BQAWA* applied to 0/1 Knapsack and TSP, with different sub-optimality bounds. The results shows for all the

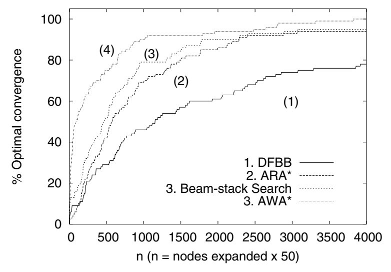

Figure 7: Percentage optimal convergence versus node expansions - 0/1 Knapsack problem

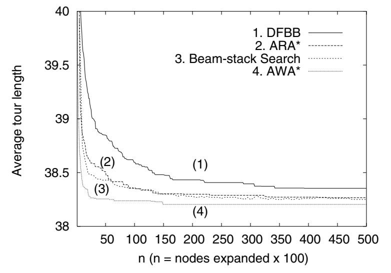

Figure 8: Average tour length versus node expansions - Euclidean TSP

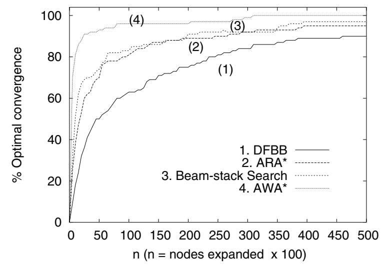

Figure 9: Percentage optimal convergence versus node expansions - Euclidean TSP

bounds, average node expansions required by BQAWA* is less than that required by ARA*, reiterating the superiority of the window based restrictions.

From the results presented, we can conclude that the AWA* algorithm provides considerably better quality-time trade-off than DFBB, ARA* or Beam-stack search. Intermediate solution qualities are better and faster optimal convergence is achieved.

## 6 Conclusions

In this paper, we proposed an anytime algorithm AWA*, which localizes the global competition performed by A* using a sliding window. Experiments performed with 0/1 Knapsack and TSP demonstrate the efficacy of AWA* over some existing anytime techniques. It may be noted, that AWA*, in its current form, is not designed to work under memory restrictions. It would be interesting to explore how the standard memory bounded techniques can be adopted in AWA*.

Table 1: Average number of node expansions with different sub-optimality bounds

| Bounds | Knapsack | | TSP | |
|---|---|---|---|---|
| | ARA* | BQAWA* | ARA* | BQAWA* |
| 2.0 | 62 | 60 | 61 | 25 |
| 1.9 | 73 | 65 | 65 | 37 |
| 1.8 | 85 | 68 | 74 | 52 |
| 1.7 | 98 | 72 | 80 | 67 |
| 1.6 | 110 | 77 | 92 | 81 |
| 1.5 | 127 | 87 | 106 | 91 |
| 1.4 | 152 | 99 | 141 | 111 |
| 1.3 | 178 | 104 | 211 | 135 |
| 1.2 | 222 | 133. | 587 | 579 |
| 1.1 | 332 | 158 | 3650 | 3585 |
| 1.0 | 59717 | 58581 | 27952 | 27905 |

## References

[Dean and Boddy, 1988] T. Dean and M. Boddy. An analysis of time-dependent planning. In *Proceedings of 6th National Conference on Artificial Intelligence (AAAI 88)*, pages 49–54, St. Paul, MN, 1988. AAAI Press.

[Furcy, 2006] D. Furcy. Itsa*: Iterative tunneling search with a*. In *Proceedings of AAAI Workshop on Heuristic Search, Memory-Based Heuristics and Their Applications*, pages 21–26, 2006.

[Hansen *et al.*, 1997] E. A. Hansen, S. Zilberstein, and V. A. Danilchenko. Anytime heuristic search: First results. Technical Report 50, Univ. of Massachusetts, 1997.

[Kitamura *et al.*, 1998] Y. Kitamura, M. Yokoo, T. Miyaji, and S. Tatsumi. Multi-state commitment search. In *Proceedings of 10th IEEE International Conference on Tools with Artificial Intelligence*, pages 431–439, 1998.

[Kumar *et al.*, 1995] A. Kumar, A. Kumar, and M. Balakrishnan. Heuristic search based approach to scheduling, allocation and binding in data path synthesis. In *Proceedings of 8th International Conference on VLSI Design*, pages 75–80, 1995.

[Likhachev *et al.*, 2004] M. Likhachev, G. J. Gordon, and S. Thrun. Ara*: Anytime A* with provable bounds on sub-optimality. In *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA, 2004.

[Pearl, 1984] J. Pearl. *Heuristics: intelligent search strategies for computer problem solving*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1984.

[Pohl, 1970] I. Pohl. Heuristic search viewed as path finding in a graph. *Artif. Intell.*, 1(3):193–204, 1970.

[Sarkar *et al.*, 1991] U. K. Sarkar, P. P. Chakrabarti, S. Ghose, and S. C. De Sarkar. Multiple stack branch and bound. *Inf. Process. Lett.*, 37(1):43–48, 1991.

[Zhang, 1998] W. Zhang. Complete anytime beam search. In *Proceedings of 14th National Conference of Artificial Intelligence AAAI'98*, pages 425–430. AAAI Press, 1998.

[Zhou and Hansen, 2002] R. Zhou and E. A. Hansen. Multiple sequence alignment using anytime a*. In *Proceedings of 18th National Conference on Artificial Intelligence AAAI'2002*, pages 975–976, 2002.

[Zhou and Hansen, 2005] R. Zhou and E. A. Hansen. Beam-stack search: Integrating backtracking with beam search. In *Proceedings of the 15th International Conference on Automated Planning and Scheduling (ICAPS-05)*, pages 90–98, Monterey, CA, 2005.