A* Search with Inconsistent Heuristics

Zhifu Zhang, Nathan R. Sturtevant,
Robert Holte, Jonathan Schaeffer
Computing Science Department
University of Alberta
Edmonton, Alberta, Canada T6G 2E8
{zfzhang, nathanst, holte, jonathan}@cs.ualberta.ca

Ariel Felner
Information Systems Engineering
Deutsche Telekom Labs
Ben-Gurion University
Be'er-Sheva, Israel 85104
felner@bgu.ac.il

Abstract

Early research in heuristic search discovered that using inconsistent heuristics with A* could result in an exponential increase in the number of node expansions. As a result, the use of inconsistent heuristics has largely disappeared from practice. Recently, inconsistent heuristics have been shown to be effective in IDA*, especially when applying the bidirectional pathmax (BPMX) enhancement. This paper presents new worst-case complexity analysis of A*'s behavior with inconsistent heuristics, discusses how BPMX can be used with A*, and gives experimental results justifying the use of inconsistent heuristics in A* searches.

1   Introduction

A* is a popular heuristic search algorithm that guarantees finding an optimal cost solution, assuming that one exists and that the heuristic used is admissible [Hart *et al.*, 1968; 1972]. However, when the heuristic is inconsistent, A* can perform very poorly as nodes that have already been expanded may need to be re-expanded many times. This results in a worst case of $O(2^N)$ node expansions, where N is the number of distinct nodes expanded [Martelli, 1977]. This motivated the creation of A* variants B [Martelli, 1977], C [Bagchi and Mahanti, 1983], and B' [Mero, 1984] with a worst-case of $O(N^2)$. Even so, these results discouraged the use of inconsistent heuristics, especially since most 'natural' heuristics seemed to be consistent (p.116, [Pearl, 1984]). However, recent research shows that there are several ways to create inconsistent heuristics such as the *dual* and *random* heuristics [Zahavi *et al.*, 2007]. In addition, any memory-based heuristic that has some values missing or degraded may be inconsistent. Inconsistency is not a problem for IDA* because IDA* already re-expands nodes many times and is only used in domains where the cost of re-expansions is fully amortized over the cost of the search. Bidirectional pathmax (BPMX) has been shown to further improve the performance of inconsistent heuristics in IDA*.

This paper complements this work by studying the applicability of using inconsistent heuristics and BPMX with A*. The following contributions are made:

- Better worst-case complexity bounds for A* that are polynomial for a large class of problems.

- Discussion of how BPMX can be integrated into A* and the resulting best- and worst-case scenarios.

- An experimental comparison of A*, B, C, B' and BPMX algorithms using a variety of types of inconsistent heuristics. The results illustrate the potential for benefit in A* searches.

2   Background

Heuristic search algorithms such as A* are guided by the cost function $f(n) = g(n) + h(n)$, where $g(n)$ is the best known distance from the initial state to state n and $h(n)$ is a heuristic function estimating the cost from n to a goal state. An *admissible heuristic* never overestimates the path cost of any node to the goal. In other words $h(n) \le h^*(n)$ for any node n [Hart *et al.*, 1968]. A *consistent heuristic* is an admissible heuristic with the property that if there is a path from node x to node y then $h(x) \le d(x,y) + h(y)$, where $d(x,y)$ is the distance from x to y [Hart *et al.*, 1968]. This is a kind of triangle inequality: the estimated distance from x to goal cannot be reduced by moving from x to a different node y and adding the estimate of the distance to goal from y to the cost of reaching y from x. Pearl [1984] showed that restricting y to be a neighbour of x produces an equivalent definition with an intuitive interpretation: in moving from a node to its neighbour h must not decrease more than g increases. If the edges in the state space are undirected, the definition of consistency can be written as $|h(x) - h(y)| \le d(x,y)$. A heuristic is *inconsistent* if it is not consistent.

A* is the algorithm of choice for many single-agent search applications. A* maintains a list of nodes to consider to be expanded (the open list) and a list of nodes that have been expanded (the closed list). The open list is sorted by increasing f-value, with ties typically being broken in favor of larger g values. At each step, the best node on the open list is moved to the closed list, expanded, and its successors are added to the open list. This continues until an optimal solution is proven. With a consistent heuristic, once a node is expanded and placed on the closed list, it never moves back to the open list.

If the heuristic is admissible and consistent, A* is "optimal" in terms of the number of node expansions [Pearl, 1984]. However, if the heuristic is admissible but not consistent, nodes can be moved back from the closed list to the open
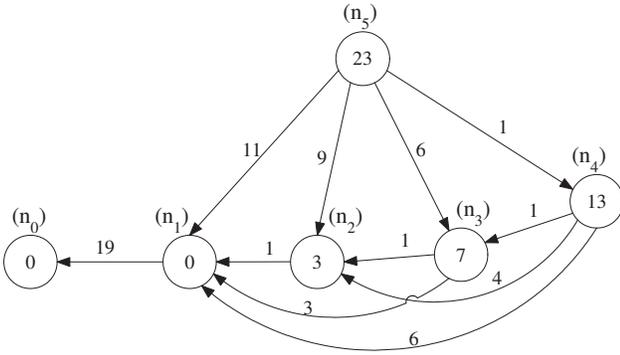
Figure 1: G_5 in Martelli's family.

Figure 2: BPMX (bidirectional pathmax).

list ("reopened") and A* can do as many as $O(2^N)$ node expansions, where N is the number of distinct expanded nodes. This was proven by Martelli [1977], who defined a family of graphs $\{G_i\}_{i=3}^{\infty}$ such that G_i contains $i + 1$ nodes and requires A* to do $O(2^i)$ node expansions to find the solution. Graph G_5 in Martelli's family is shown in Figure 1; the number inside a node is its heuristic value. There are many inconsistencies in this graph. For example, $d(n_4, n_3) = 1$ but $h(n_4) - h(n_3) = 6$. The unique optimal path from start (n_5) to goal (n_0) visits the nodes in decreasing order of their index $(n_5, n_4, ..., n_0)$, but n_4 has a large enough heuristic value $(f(n_4) = 14)$ that it will not be expanded by A* until all possible paths to the goal (with $f < 14$) involving all the other nodes have been fully explored. Thus, when n_4 is expanded, nodes n_3, n_2 and n_1 are reopened and then expanded again. Moreover, once n_4 is expanded, the same property holds again of n_3, the next node on the optimal path, so it is not expanded until all paths from n_4 to the goal involving all the other nodes have been fully explored. This pathological pattern of behavior repeats each time one additional node on the optimal path is expanded for the last time. As we will show below this worst-case behavior hinges on the search graph having the properties, clearly seen in the definition of Martelli's family, that the edge weights and heuristic values grow exponentially with the graph size.

Martelli [1977] devised a variant of A*, called B, that improves upon A*'s worst-case time complexity while maintaining admissibility. Algorithm B maintains a global variable F that keeps track of the maximum f-value of the nodes expanded so far. When choosing the next node to expand, if f_m, the minimum f-value in the open list, satisfies $f_m \geq F$, then the node with minimum f-value is chosen as in A*, otherwise the node with minimum g-value among those with $f < F$ is chosen. Because the value of F can only change (increase) when a node is expanded for the first time, and no node will be expanded more than once for a given value of F, the worst-case time complexity of algorithm B is $O(N^2)$.

Bagchi and Mahanti [1983] proposed C, a variant of B, by changing the condition for the special case from $f_m < F$ to $f_m \leq F$ and altering the tie-breaking rule to prefer smaller g values. C's worst-case time complexity is the same as B's, $O(N^2)$.
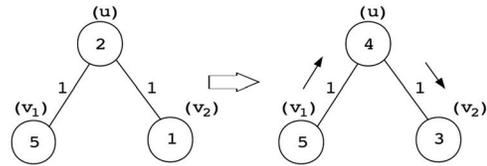
Mero [1984] modified B to create B′ by introducing two "pathmax" rules that propagate heuristic values between a parent node n and its successor m during search as follows:

(a) For each successor m of the selected node n, if $h(m) < h(n) - d(n, m)$, then set $h(m) \leftarrow h(n) - d(n, m)$.

(b) Let m be the successor node of n for which $h(m) + d(n, m)$ is minimal. If $h(n) < h(m) + d(n, m)$, then set $h(n) \leftarrow h(m) + d(n, m)$.[1]

Rule (a) updates the successors' heuristic values, and (b) updates the parent's heuristic value. Like B, B′ has a worst-case time complexity of $O(N^2)$.

Bidirectional pathmax (BPMX) [Felner *et al.*, 2005] is a method that works with inconsistent heuristics and propagates large values to neighboring nodes. It can be seen as applying Mero's pathmax rule (a) in both directions when the edge connecting the two nodes is undirected. This is illustrated in Figure 2, where u is the node being expanded, nodes v_1 and v_2 are its two neighbors, and the number in a node is its h value. $h(v_1)$ can propagate to u, updating its value to 4 (5 - $d(u, v_1) = 4$). In turn, $h(u)$ can propagate to v_2, updating its value to 3 (4 - $d(u, v_2) = 3$). All previous research on BPMX has been in the context of IDA*, not A*. In IDA* BPMX propagation is essentially "free" computationally, because it can be done as part of the backtracking that is intrinsic to the IDA* search. If the IDA* search threshold is, for example, 3 and u is at the root of the search tree then having searched v_1, the backed up value of u becomes 4 causing a cut-off and child v_2 is not explored. Section 4 below points out that only very limited versions of BPMX can be added to A* for "free", and discusses the costs and benefits of using more complete versions of BPMX in A*.

3 Worst-Case Complexity Analysis

Although Martelli proved that the number of node expansions A* performs may be exponential in the number of distinct nodes expanded, this behavior has never been reported in real-world applications of A*. His family of worst-case graphs have edge weights and heuristic values that grow exponentially with the graph size. We show here that these are necessary conditions for A*'s worst-case behavior to occur.

Let V be the set of nodes expanded by A* and $N = |V|$. We assume all edge weights are positive integers. The key quantity in our analysis is Δ, defined to be the greatest common divisor of all the edge weights. The cost of every path from the start node to node n is a multiple of Δ, and so too

---

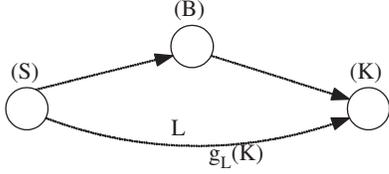[1]This is our version of the second pathmax rule. The version in [Mero, 1984] is clearly not correct.

Figure 3: First and last explored path.

is the difference in the costs of any two paths from the start node to n. Therefore, if during search we reopen n because a new path to it is found with a smaller cost than our current $g(n)$ value, we know that $g(n)$ will be reduced by at least Δ.

Theorem 1 If A* performs $\phi(N) > N$ node expansions then there must be a node with heuristic value of at least $LB = \Delta * \lceil (\phi(N) - N)/N \rceil$.
Proof. If there are $\phi(N)$ total expansions by A*, then the number of re-expansions is $\phi(N) - N$. By the pigeonhole principle there must be a node, say K, with at least $\lceil (\phi(N) - N)/N \rceil$ re-expansions. Each re-expansion must decrease $g(K)$ by at least Δ, so after this process the g-value of K is reduced by at least $LB = \Delta * \lceil (\phi(N) - N)/N \rceil$.

In Figure 3, S is the start node, K is any node that is re-expanded at least $\lceil (\phi(N) - N)/N \rceil$ times (as we have just seen, at least one such node must exist), the lower path to K, L, is the path that resulted in the first expansion of K, and the upper path to K (via node B) is the path that resulted in the last expansion of K. We denote the f- and g-values along path L as f_L and g_L, and the f- and g-values along the upper path as f_{last} and g_{last}, respectively.

Node B is any node on the upper path, excluding S, with the maximum f_{last} value. Nodes distinct from S and K must exist along this path because if it were a direct edge from S to K, K would be open as soon as S was expanded with a g-value smaller than $g_L(K)$ so K would not be expanded via L, a contradiction. Node B must be one of these intermediate nodes — it cannot be S by definition and it cannot be K because if $f_{last}(K)$ was the largest f_{last} value, the entire upper path would be expanded before K would be expanded via L, again a contradiction. Hence, B is an intermediate node between S and K.

$h(B)$ must be large enough to make $f_{last}(B) \geq f_L(K)$ (because K is first expanded via L). We will now use the following facts to show that $h(B)$ must be at least LB:

$$f_{last}(B) = g_{last}(B) + h(B) \qquad (1)$$
$$f_{last}(B) \geq f_L(K) \qquad (2)$$
$$f_L(K) = g_L(K) + h(K) \qquad (3)$$
$$g_{last}(B) < g_{last}(K) \qquad (4)$$
$$LB \leq g_L(K) - g_{last}(K) \qquad (5)$$

So,

$$
\begin{aligned}
h(B) &= f_{last}(B) - g_{last}(B), \text{ by Fact 1}\\
&\geq f_L(K) - g_{last}(B), \text{ by Fact 2}\\
&= g_L(K) + h(K) - g_{last}(B), \text{ by Fact 3}\\
&> g_L(K) - g_{last}(K) + h(K), \text{ by Fact 4}\\
&\geq g_L(K) - g_{last}(K), \text{ since } h(K) \geq 0\\
&\geq LB, \text{ by Fact 5 } \square
\end{aligned}
$$

From Theorem 1 it follows that for A* to expand 2^N nodes, there must be a node with heuristic value of at least $\Delta * \lceil (2^N - N)/N \rceil$, and for A* to expand N^2 nodes, there must be a node with heuristic value of at least $\Delta * (N - 1)$.

Corollary 1 Let $g^*(goal)$ denote the optimal solution cost. If A* performs $\phi(N) > N$ node expansions then $g^*(goal) \geq LB$.
Proof. Since A* expanded node B before the goal, $g^*(goal)$ must be at least $f(B)$, which is at least LB. $\square$

Corollary 2 If $g^*(goal) \leq \lambda(N)$, then $\phi(N) \leq N + N * \lambda(N)/\Delta$.
Proof. Using Corollary 1,
$$\Delta * \lceil (\phi(N) - N)/N \rceil = LB \leq g^*(goal) \leq \lambda(N)$$
which implies
$$\phi(N) \leq N + N * \lambda(N)/\Delta \qquad \square$$

Corollary 3 Let m be a fixed constant and G a graph of arbitrary size (not depending on m) whose edge weights are all less than or equal to m. If N is the number of nodes expanded by A* when searching on G then the total number of node expansions by A* during this search is at most $N + N * m * (N - 1)/\Delta$.
Proof. Because the non-goal nodes on the solution path must each have been expanded, there are at most $N-1$ edges in the solution path and $g^*(goal)$ is therefore at most $m * (N - 1)$. Using Corollary 2,
$$\phi(N) \leq N + N * \lambda(N)/\Delta \leq N + N * m * (N - 1)/\Delta \square$$

This shows that, when a graph's edge weights do not depend on its size, A* does not have an asymptotic disadvantage compared to B, C, and B'; all have a worst-case time complexity of $O(N^2)$. Using A* with inconsistent heuristics under these common conditions has a much better time complexity upper bound than previously thought. For example, if the graph is a square $L \times L$ grid with unit edge weights, then $N \leq L^2$, the optimal solution path cost is at most $2\sqrt{N}$, and the worst-case time complexity of A* using inconsistent heuristics is $O(N^{\frac{3}{2}})$. For many problems the optimal solution cost grows asymptotically slower than N, such as $\ln(N)$. Here A* has a worst-case complexity that is better than $O(N^2/\Delta)$

4 BPMX in A*

BPMX is easy to implement in IDA* as part of the normal search procedure. As IDA* does not usually keep all successors of a state in memory simultaneously; heuristic values are only propagated by BPMX to unexpanded children and never back to previously expanded children as they have already been fully explored. But, in A* all successors are generated and processed before other expansions occur, which means that in A* BPMX should be implemented differently.

We parameterize BPMX with the amount of propagation. BPMX(∞) is at one extreme, propagating h updates as far as possible. BPMX(1) is at the other extreme, propagating h updates only between a node and its immediate neighbors. In general, there are four possible overheads associated with BPMX within the context of A*:
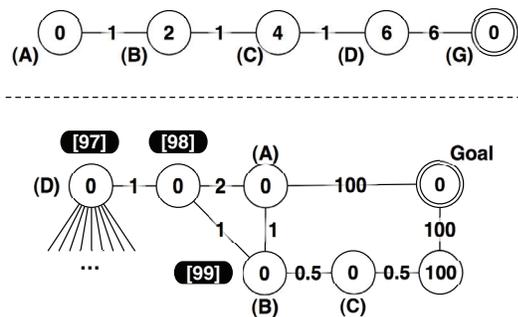
(a) performing lookups in the open and/or closed lists,

Figure 4: Good and bad examples for BPMX.

(b) ordering open list nodes based on their new f-value,
(c) moving closed nodes to open (reopening), and
(d) computational overhead.

BPMX(1) with A* works as follows. Assume that a node p is expanded and that its k children $v_1, v_2, \ldots, v_k$ are generated, which requires a lookup in the open and/or closed lists. All these nodes are then at hand and are easily manipulated. Let v_{max} be the node with the maximum heuristic among all the children and let $h_{max} = h(v_{max})$. Assuming that each edge has a unit cost, we can now propagate h_{max} to the parent node by decreasing h_{max} by one and then to the other children by decreasing it by one again. A second update is required to further propagate any updated values and then to write them to the open or closed list.

In A* the immediate application of pathmax is 'free', as it only requires an additional 'max' calculation, but BPMX has additional overhead. BPMX(1) can be implemented efficiently if the expansion of successors is broken into generation and processing stages, with the BPMX computation happening after all successors have been generated and retrieved from the open or closed list, but before changes have been written back out to the relevant data structures. BPMX(d) with $d > 1$ requires performing a small search, propagating heuristic values to nodes that are not initially in memory.

No fixed BPMX propagation policy is optimal for all graphs. While a particular propagation policy can lead, in the best case, to large savings, on a different graph it can lead to a $O(N^2)$ increase in the number of nodes expanded.

Figure 4 (top) gives an example of the worst-case behavior of BPMX(∞) propagation. The heuristic values gradually increase from nodes A to G. When node B is reached, the heuristic can be propagated back to node A, increasing the heuristic value to 1. When node C is reached, the heuristic update can again be propagated back to nodes B and A. In general, when the ith node in the chain is generated a BPMX update can be propagated to all previously expanded nodes. Overall this will result in $1 + 2 + 3 + \cdots + N - 1 = O(N^2)$ propagation steps with no savings in node expansions. This provides a general worst-case bound. At most, the entire set of previously expanded nodes can be re-visited during BPMX propagations, which is what happens here. But, in this example BPMX(1) has no asymptotic overhead.

By contrast, Figure 4 (bottom) gives an example of how full BPMX propagation can be very effective. The start node is A. The search proceeds to node C which has a child with heuristic value of 100. After a BPMX(1) update, $f(C) = 101$

and $h(B) = 99$, so node expansions will continue at node D, where an arbitrary large number of nodes can be expanded. With BPMX(∞), updates will continue until $h(D) = 97$ and $f(D) = 100$. At this point the optimal path to the goal will be expanded before any children of D. By adding extra nodes between B and C (with lower edge costs), an arbitrary large parameter for BPMX can be required to see these savings.

5 Experiments

We now have four algorithms (A*, B, B', and C) that all have similar asymptotic worst-case complexity if applied with inconsistent heuristics to the grid-like search spaces that are found in computer video game applications as well as other domains. In addition, we have A* augmented with BPMX(r), for any propagation distance r. In this section we compare these algorithms experimentally with a variety of (in)consistent heuristics. We experimented with BPMX(r) for $r \in \{1, 2, 3, \infty\}$.

All experiments are performed on Intel P4 computers (3.4GHz) with 1GB of memory and use search spaces that are square grids in which each non-border cell has eight neighbours—4 cardinal (distance = 1) and 4 diagonal (distance = $\sqrt{2}$). Octile distance is an easy-to-compute consistent heuristic in this domain. If the distances along x and y coordinates between two points are (dx, dy), then the octile distance between them is $\sqrt{2} * min(dx, dy) + |dx - dy|$.

Each algorithm is run on the same set of start/goal instances, which are divided into buckets based on their solution lengths (from 5 to 512). Each bucket contains the same number of randomly generated start/goal instances.

The experiments differ in how the inconsistent heuristics were created. The first experiment uses a realistic method to create inconsistency. The final two experiments use artificial methods to create inconsistency in a controlled manner.

5.1 Random Selection From Consistent Heuristics

In this experiment our search spaces are a set of 116 maps from commercial games, all scaled to be 512 by 512 × size. There are blank spots and obstacles on the maps. There are 128 test instance buckets, each containing 1160 randomly-generated problem instances. In this section BPMX refers to BPMX(1) which performed best in this domain.

We generate an inconsistent heuristic similar to [Zahavi et al., 2007] by maintaining a certain number, H, of differential heuristics [Sturtevant et al., 2009], each formed by computing shortest paths to all points in the map from a random point t. Then, for any two points a and b, $h(a, b) = |d(a, t) - d(b, t)|$ is a consistent heuristic. To compute a heuristic for node n we systematically choose just one of the H heuristics to consult. Inconsistency is almost certain to arise because different heuristics will be consulted for a node and its children. We take the maximum of the result with the default octile heuristic, and call the result the enhanced octile heuristic. By design, the enhanced octile heuristic dominates the octile heuristic. The enhanced octile heuristic is higher than the octile heuristic in roughly 25% of the nodes.

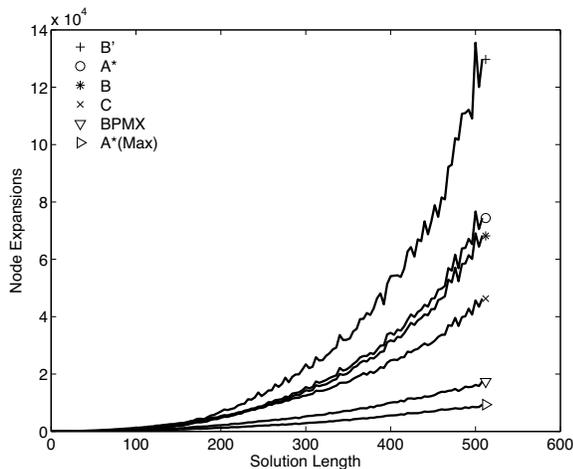The number of node expansions by algorithms A*, B, C, B', and A* with BPMX when using the inconsistent heuris-

Figure 5: Node expansions with random selection of differential heuristics.

tic are plotted in Figure 5. A* using the maximum of all the heuristics ("A*(Max)") is plotted for reference. The x-axis is the solution length, the y-axis is the number of node expansions. The legend is in the same descending order as the lines for the algorithms. When counting node expansions, a BPMX propagation from a child to its parent is counted as an additional expansion ("reverse expansion").

As can be seen, B′ does the most node expansions, and, as expected, A*(Max) does the fewest. Among the lines using the inconsistent heuristic, A* with BPMX is best and is within a factor of two of A*(Max). As the number of available heuristics grows, A*(Max) will have increasing running time, while the inconsistent heuristic will not.

An unexpected result is that B′ expands more nodes than B. This is contrary to a theoretical claim in [Mero, 1984]. This discrepancy is a result of tie-breaking rules. B′ would have the same performance as B (or better) if, when faced with a tie, it could choose the same node to expand as B. But, in practice this isn't feasible. The pathmax rules in B′ cause many more nodes to have the same f-cost than when searching with B. When breaking ties between these nodes, B′ is unable to infer how B would break these ties, and thus has different performance. If we had a tie-breaking oracle, we expect B′ and B would perform similarly.

Detailed analysis of the data for the hardest bucket of problems is shown in Table 1. Column "First" is the number of distinct nodes expanded, "Re-Exp" is the number of node re-expansions, "BMPX" is the number of BPMX reverse expansions, and "Sum" is the sum of those three columns, the total number of node expansions. "Time" is the average CPU time, in seconds, needed to solve one instance. A*(Max) is the best but its time advantage is less because it performs multiple heuristic lookups per node. Algorithm B uses slightly more time than A*, despite fewer node expansions. This is because B occasionally needs to extract the node with minimum g value from the open list, which is sorted by f. B′ expands approximately the same number of distinct nodes as A* and B, but B′ performs many more re-expansions. BPMX

Alg.	First	Re-Exp	BPMX	Sum	Time
A*(Max)	9341	0	0	9341	0.066
A*	17210	57183	0	74392	0.503
B	17188	50963	0	68151	0.560
B′	16660	112010	0	129680	0.717
C	21510	24778	0	46288	0.411
BPMX	10195	4065	3108	17368	0.089
BPMX(2)	9979	3462	5545	18986	0.093
BPMX(3)	9997	3467	5854	19317	0.094
BPMX(∞)	10025	3483	6207	19714	0.094

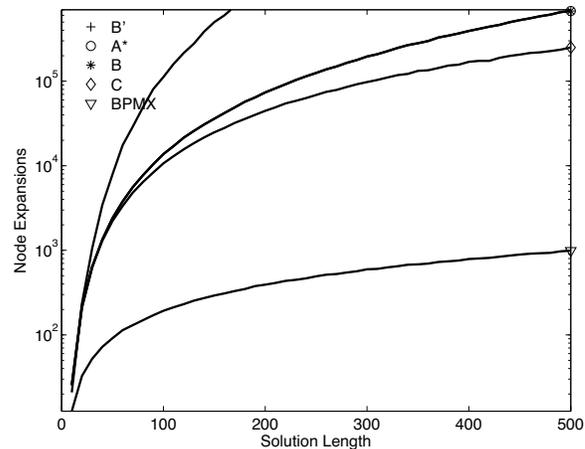Table 1: Last bucket in differential heuristic experiment.

Figure 6: Perfect heuristics ($p = 0.5$).

is able to dramatically reduce the number of distinct nodes expanded and re-expansions at the cost of a few reverse expansions. The number of distinct nodes expanded by BPMX is close to that of A*(Max). The last four rows show there is little difference between the BPMX variants in terms of nodes and average execution time, although increasing the propagation parameter increases the number of propagations.

5.2 Inconsistency by Degrading Perfect Heuristics

To test the generality of the preceding results, we have created inconsistent heuristics by degrading exact distances (perfect heuristic values). We do this not for performance, but in order to compare the algorithms with various types of heuristics. The grids in the two experiments in this section were all 1000×1000 in size and obstacle free, and the test instances were divided into 50 buckets, with each bucket containing 1,000 randomly generated start/goal instances.

In the first experiment, each node has a perfect heuristic value (the exact distance to goal) with probability p, and has a heuristic value of 0 otherwise. We experimented with $p = 0.25$ and $p = 0.5$. This experiment is favorable to BPMX(1) because with very high probability, either a node or one its neighbours will have a perfect heuristic value, which BPMX(1) will then propagate to the other neighbors.

Figure 6 shows the number of node expansions as a function of solution length (bucket) for $p = 0.5$; the plot for $p = 0.25$ is similar. The y-axis is a log scale. For both values

638

Alg.	First	Re-Exp	BPMX	Sum	Time
A*	175146	501043	0	676190	2.9634
B	175146	501043	0	676190	4.5587
C	197378	55161	0	252539	2.7401
BPMX(1)	650	0	340	991	0.0048
BPMX(2)	650	0	340	991	0.0035
BPMX(3)	650	0	340	991	0.0034
BPMX(∞)	650	0	340	991	0.0033

Table 2: Perfect heuristics ($p = 0.5$, hardest cases).

of p the same pattern is seen: B$'$ does many more expansions than any other algorithm, A*, B, and C do roughly the same number of node expansions, and A* with BPMX(1), as expected, does over two orders of magnitude fewer node expansions. This is an example of best-case performance for BPMX. B has a larger running time (not shown) due to its more complicated data structures.

Table 2 examines the algorithms' performance with $p = 0.5$ on the instances with the longest solutions in more detail. B$'$ is omitted from the table because it could not solve the largest problems in reasonable amounts of time. The propagation parameter for BPMX does not matter in these experiments, because good heuristic values are always close by. The same pattern is seen for $p = 0.25$.

Our second experiment investigated the behavior of the algorithms when the heuristic is locally consistent but globally inconsistent. We overlay a coarse-grained grid on the 1000×1000 search space and imagine the overlay coloured in a checkerboard fashion. If a node lies in a white section of the coarse-grained grid, its heuristic is perfect; otherwise its heuristic value is 0.

We present the results on grid overlays of width 10 (Table 3) and 50 (Table 4). As the grid overlay gets larger, larger values for BPMX propagation perform better. This is because BPMX is able to push updates from the borders of the grid farther back into the OPEN list and therefore avoid additional expansions. Thus, we see that BPMX shows good promise in practically reducing node expansions, and that the worst-case is unlikely to occur in practice.

6 Conclusions

This research makes the case that inconsistent heuristics are not as bad for A* as previously thought. In particular, for many problems, the worst-case bound is $O(N^2)$ instead of

$O(2^N)$. When A* does have poor performance, BPMX is able to markedly improve the performance of A* search with inconsistent heuristics. Although BPMX has the same worst-case as A*, that worst-case does not seem to occur in practice.

As pointed out in [Zahavi *et al.*, 2007] there are several easy ways to create inconsistent heuristics. Combined with the case already made for IDA* search, the results in this paper encourage researchers and application developers to explore inconsistency as a means to further improve the performance of search with A* and similar algorithms.

References

[Bagchi and Mahanti, 1983] Amitava Bagchi and Ambuj Mahanti. Search Algorithms Under Different Kinds of Heuristics-A Comparative Study. *Journal of the ACM*, 30(1):1–21, 1983.

[Felner *et al.*, 2005] Ariel Felner, Uzi Zahavi, Jonathan Schaeffer, and Robert Holte. Dual Lookups in Pattern Databases. In *IJCAI*, pages 103–108, 2005.

[Hart *et al.*, 1968] Peter Hart, Nils Nilsson, and Bertram Raphael. A Formal Basis for the Heuristic Determination of Minimum-Cost Paths. *IEEE Transactions of Systems Science and Cybernetics*, SSC-4(2):100–107, 1968.

[Hart *et al.*, 1972] Peter Hart, Nils Nilsson, and Bertram Raphael. Correction to "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". *SIGART Newsletter*, 37:28–29, 1972.

[Martelli, 1977] Alberto Martelli. On the Complexity of Admissible Search Algorithms. *Artificial Intelligence*, 8(1):1–13, 1977.

[Mero, 1984] Laszlo Mero. A Heuristic Search Algorithm with Modifiable Estimate. *Artificial Intelligence*, 23(1):13–27, 1984.

[Pearl, 1984] Judea Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley, 1984.

[Sturtevant *et al.*, 2009] Nathan Sturtevant, Ariel Felner, Max Barer, Jonathan Schaeffer, and Neil Burch. Memory-Based Heuristics for Explicit State Spaces. In *IJCAI*, 2009.

[Zahavi *et al.*, 2007] Uzi Zahavi, Ariel Felner, Jonathan Schaeffer, and Nathan Sturtevant. Inconsistent Heuristics. In *AAAI*, pages 1211–1216, 2007.

Alg.	First	Re-Exp	BPMX	Sum	Time
A*	210271	106289	0	316561	3.0812
B	210271	106289	0	316561	5.2529
C	220920	45482	0	266403	5.5113
BPMX(1)	625	4	286	915	0.0061
BPMX(2)	618	3	287	910	0.0078
BPMX(3)	617	3	287	908	0.0076
BPMX(∞)	616	3	285	905	0.0079

Table 3: Hardest Problems in Perfect Heuristic Checkerboard Experiment. Gridwidth=10

Alg.	First	Re-Exp	BPMX	Sum	Time
A*	208219	26665	0	234884	2.0505
B	208219	26665	0	234884	4.2371
C	220920	45482	0	266403	5.5113
BPMX(1)	1389	438	810	2638	0.0613
BPMX(2)	1049	122	640	1811	0.0157
BPMX(3)	1046	119	660	1825	0.0165
BPMX(∞)	1023	114	650	1788	0.0151

Table 4: Hardest Problems in Perfect Heuristic Checkerboard Experiment. Gridwidth=50