Common Misconceptions Concerning Heuristic Search

Robert C. Holte
Computing Science Department, University of Alberta
Edmonton, Canada T6G 2E8
(holte@cs.ualberta.ca)

Abstract

This paper examines the following statements about heuristic search, which are commonly held to be true:

- More accurate heuristics result in fewer node expansions by A* and IDA*.
- A* does fewer node expansions than any other equally informed algorithm that finds optimal solutions.
- Any admissible heuristic can be turned into a consistent heuristic by a simple technique called $pathmax$.
- In search spaces whose operators all have the same cost A* with the heuristic function $h(s) = 0$ for all states, s, is the same as breadth-first search.
- Bidirectional A* stops when the forward and backward search frontiers meet.

The paper demonstrates that all these statements are false and provides alternative statements that are true.

Introduction

Heuristic search is one of the pillars of Artificial Intelligence. Sound knowledge of its fundamental results and algorithms, A* (Hart, Nilsson, & Raphael 1968) and IDA* (Korf 1985), is requisite knowledge for all AI scientists and practitioners. Although its basics are generally well understood, certain misconceptions concerning heuristic search are widespread. In particular, consider the following assertions about heuristic search:

- If admissible heuristic h_2 is more accurate than admissible heuristic h_1 A* and IDA* will do fewer node expansions if they use h_2 than if they use h_1.
- A* is optimal, in the sense of doing fewer node expansions than any other equally informed algorithm that finds optimal solutions.
- Any admissible heuristic can be turned into a consistent heuristic by a simple technique called $pathmax$.
- In search spaces whose operators all have the same cost A* with the heuristic function $h(s) = 0$ for all states, s, is the same as breadth-first search.
- Bidirectional A* stops when the forward and backward search frontiers meet.

Although these statements are intuitively highly plausible and are widely held to be true, as a matter of fact, they are all false. The aim of this paper is to demonstrate how they fail with simple counterexamples and to provide alternatives to these statements that are true. The falsity of the above statements, and the corrections given, have all have been reported previously, but usually in specialized publications. The contributions of this paper are to draw attention to them, to bring them together in one widely accessible document, and to give simple counterexamples.

Background, Terminology, and Notation

This section briefly reviews the terminology, notation, and essential facts needed in the remainder of the paper. It is not a full tutorial on heuristic search.

A state space consists of a set of states, a successor relation defining adjacency between states, and a function defining the cost of moving from state s to adjacent state t.

A* and IDA* are algorithms for finding a best (least-cost) path from any given state, $start$, to a predetermined goal state, $goal$. Both make use of three functions, g, h and f. $g(s)$ is the cost of the best known path from $start$ to state s at the current stage of the search. $h(s)$, the heuristic function, estimates the cost of a best path from state s to $goal$. $f(s) = g(s) + h(s)$ is the current estimate of the minimum cost of reaching $goal$ from $start$ with a path passing through s. The true minimum cost of a path from $start$ to $goal$ is denoted $f*$.

All heuristic functions are non-negative and have $h(goal) = 0$. Heuristic $h(s)$ is admissible if, for every state s, it does not overestimate the cost of a best path from s to $goal$. A* and IDA* are guaranteed to find a least-cost path from $start$ to $goal$ if $h(s)$ is admissible. $h(s)$ is consistent (p. 83, (Pearl 1984)) if for every two states[1], s and t,

$$h(s) \leq cost(s,t) + h(t) \quad (1)$$

where $cost(s,t)$ is the cost of a least-cost path from s to t. A consistent heuristic is guaranteed to be admissible.

A* maintains a list of states called $OPEN$. On each step of its search A* removes a state in $OPEN$ with the smallest

[1] Pearl (1984) showed that restricting t to be a neighbour of s produces an equivalent definition that is easier to verify in practice and has an intuitive interpretation: in moving from a state to its neighbour h must not decrease more than g increases.

f-value and "expands" it, which means marking the state as "closed", computing its successors, and putting each successor in $OPEN$ if it has not previously been generated or if this path to it is better than any previously computed path to it. A* terminates as soon as $goal$ has the smallest f-value in $OPEN$. When A* is executed with a consistent heuristic, the f-values of the states it expands as search progresses form a monotone non-decreasing sequence. This is not true, in general, if A* is executed with an inconsistent heuristic.

IDA* does a series of cost-bounded depth-first searches. When searching with a specific cost bound θ, state s is ignored if $f(s) > \theta$. If $f(s) \leq \theta$ the successors of s are searched in a depth-first manner with the same cost bound. IDA* terminates successfully as soon as $goal$ is reached by a path whose cost is less than or equal to θ. If search does not terminate successfully with the current value of θ, the smallest f-value exceeding θ seen during the current iteration is used as θ's new value for a depth-first search that begins afresh from $start$.

The basic operation of both A* and IDA* is to expand one state, and the time complexity of the algorithms is measured by the number of times this operation is executed, *i.e.*, the number of state expansions (or, as they are more commonly called, "node expansions").

Better Heuristics Can Result in More Search

One admissible heuristic, h_2, is defined to be "better than" (or "dominate") another, h_1, if for all states s, $h_1(s) \leq h_2(s)$ and there exist one or more states s for which $h_1(s) < h_2(s)$. Is A* guaranteed to do fewer node expansions when it is given the better heuristic ?

Although intuition urges the answer "yes", and there do exist provable connections between the accuracy of a heuristic and the number of node expansions A* will do (Dinh, Russell, & Su 2007), the true answer is much more complex (pp. 81-85, (Pearl 1984)). Even when the heuristics are consistent, the answer is not an unequivocal "yes" because with the better heuristic A* might expand arbitrarily more states that have $f(s) = f*$.

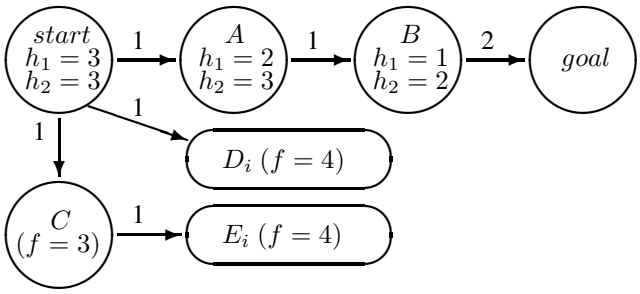

Figure 1: The better heuristic (h_2) results in more A* search.

Consider the state space in Figure 1, and two consistent heuristics, h_1 and h_2, which are identical except along the optimal path, $start - A - B - goal$. h_2 is better than h_1. The ovals labelled D_i and E_i represent arbitrarily large sets of states with $f = 4$. States in D_i are successors of $start$ with $g = 1$. States in E_i are successors of C and have $g = 2$.

Using h_1, A* expands four states. First it will expand $start$, then A, B and C in some order (all have f=3). At this point search will terminate. None of the states in sets D_i or E_i will be expanded because as soon as all states with $f < 4$ are expanded, $goal$ is in the $OPEN$ list with $f = 4$ so search terminates.

By contrast, using h_2 there is no systematic way for A* to avoid expanding some of the states in D_i or E_i. After $start$ and C (f=3) are expanded, the $OPEN$ list contains a large set of states with $f = 4$, but does not contain $goal$. If A* breaks ties in favour of larger g values, as is most often done, all the states in E_i will be expanded before A will be expanded. If A* breaks ties in the opposite manner, all the states in D_i will be expanded before B will be expanded. If A* breaks ties without considering g, it has no way of identifying A and B as being preferable over the states in D_i and E_i, and therefore cannot guarantee that A and B will be expanded before any of the states in D_i and E_i.

The key point in this example is that low heuristic values are not always harmful. If they occur along an optimal path low heuristic values are beneficial because they will cause the goal to be placed in $OPEN$ earlier, potentially reducing the number of states with $f = f*$ that are expanded.

There is an additional source of confusion on this topic when the heuristics are admissible but not consistent. In this case, A* may have to expand the same state many times (see Figure 3 below and the text discussing it). Some theorems do not take into account repeated expansions of the same state, they only show that under certain conditions the set of states expanded using h_1 is a superset of the set expanded using h_2.

What about IDA* ?

The preceding counterexample does not apply to the standard implementations of IDA* because they fail to notice that during the iteration with $\theta = 3$ $goal$ was encountered with $f = 4$. If IDA* was programmed to notice this, it could terminate as soon as θ was increased to 4 and save considerable effort. The savings would only occur if h_1 was used. If the better heuristic, h_2, were used, at least part of the $\theta = 4$ iteration would have to be performed.

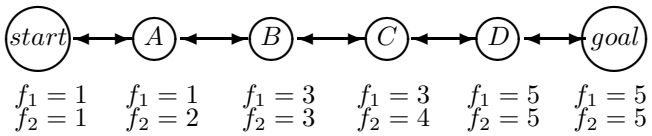

Figure 2: The better heuristic (h_2) requires more IDA* iterations.

An example of a better heuristic leading to more search with a standard IDA* implementation is shown in Figure 2. h_1 alternates between 1 (for $start, B$, and D) and 0 (for A, C, and $goal$). h_2 is 1 on every state except $goal$, and is therefore better than h_1. Both heuristics are consistent. The f-values corresponding to these two heuristics are shown in the figure. The key point is that h_1 produces only three

distinct f-values, whereas the better heuristic produces five. This means IDA* will do two extra iterations with h_2 and, consequently, will do more node expansions using h_2 than h_1. Although rare, this does arise in practice. In one case (Holte *et al.* 2006) use of a better heuristic increased the number of node expansions by IDA* by more than 50%.

This phenomenon was first noted in Manzini's comparison of the perimeter search algorithm BIDA* with IDA* (Manzini 1995). Manzini (p. 352) observed that BIDA* cannot do more node expansions than IDA* for a given θ but that BIDA* can do more node expansions than IDA* in total because "the two algorithms [may] execute different iterations using different thresholds".

A* is Not Always Optimal

There is a general belief that A* is optimal among search algorithms that use the same information, in the sense of doing the fewest possible node expansions and still being guaranteed to find an optimal solution path. This claim has been a source of confusion throughout its history (p. 111, (Pearl 1984)).

This claim is certainly false when the heuristic being used is admissible but not consistent. In this case, A* may have to move closed states back onto $OPEN$ where they might be expanded again. In the worst case, A* can do as many as $O(2^N)$ node expansions, where N is the number of distinct states that are expanded in solving the problem, while there are rival search algorithms[2] that do only $O(N^2)$ node expansions in the worst case. This was proven by Martelli (1977), who defined a family of graphs G_i, for $i \geq 3$, such that G_i contains $i+1$ states and A* does $O(2^i)$ node expansions to find the solution. Graph G_5 in Martelli's family is shown in Figure 3.[3] In this figure, the value inside a circle (state) is the state's heuristic value. There are many inconsistencies in this graph. For example, $d(n_4, n_3) = 1$ but $h(n_3)$ is 6 smaller than $h(n_4)$. The unique optimal path from $start$ (n_5) to $goal$ (n_0) visits the states 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 states have been fully explored. Thus, when n_4 is expanded, states 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 state on the optimal path, so it is not expanded until all paths from n_4 to the goal involving all the other states have been fully explored. This pathological pattern of behavior repeats each time one additional state on the optimal path is expanded for the last time.

Martelli (1977) also introduced 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 seen so far among the states expanded. When choosing the next state to expand, if the minimum f-value in the open list (denoted f_m) satisfies $f_m \geq F$, then a state with minimum f is chosen as in A*, otherwise a state with minimum g-value among the states with $f < F$ is chosen for expansion. Because F can only change (increase) when a state is expanded for the first time, and no state will be expanded more than once for a given value of F, the worst-case time complexity of algorithm B is $O(N^2)$. Other A* variants with $O(N^2)$ worst-cases have also been developed (Bagchi & Mahanti 1983; Mero 1984). A description and experimental comparison of these algorithms can be found in (Zhang *et al.* 2009), which also shows that A*'s exponential worst-case requires the solution costs and heuristic values to grow exponentially with N.

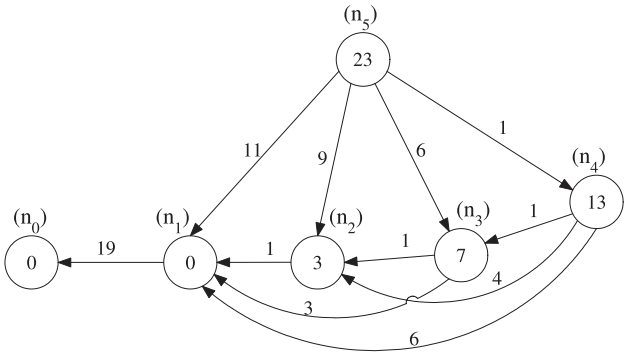

Figure 3: G_5 in Martelli's family.

This example shows that, when the heuristic being used is admissible but not consistent, A* can do exponentially more node expansions than its rivals. In fact, it can be shown that there is no optimal search algorithm for admissible, inconsistent heuristics (Dechter & Pearl 1983). The situation is different when the heuristic is consistent. In this case A* "largely dominates" its rivals, which means that A* does not do more node expansions than any of its rivals, except perhaps for some states with $f(s) = f*$ (p. 85, (Pearl 1984); Theorem 3 in (Dechter & Pearl 1983)).

In discussing the optimality of A* it is important to bear in mind that the algorithm that does the fewest node expansions is not necessarily the algorithm that runs fastest (Korf 1985). For example, the $Fringe$ algorithm introduced in (Bjornsson *et al.* 2005) does more node expansions than A* in the experiments reported but runs 25-40% faster, even though the data structures used in the A* implementation were highly optimized.

Pathmax does not make Heuristics Consistent

It is commonly understood that there is a simple method, called $pathmax$, to turn any admissible, inconsistent heuristic into a consistent one. $Pathmax$ is based on the idea that when state s is reached by A* by some path, it is admissible to use the maximum f-value seen along the path instead of $f(s)$.

While it is true, trivially, that with $pathmax$ f-values

[2] A*'s "rivals" are other search algorithms that use the same information and return an optimal solution when given an admissible heuristic.

[3] This figure and the related text are copied from (Zhang *et al.* 2009).

never decrease along a path, this is not the same as the heuristic being converted into a $bonafide$ consistent heuristic. To see this, recall that with a consistent heuristic, closed states are never re-opened by A*, because when a state is removed from $OPEN$ for the first time we are guaranteed to have found the least-cost path to it. This is the key advantage of a consistent heuristic over an inconsistent, admissible heuristic. $Pathmax$ does not correct this deficiency of inconsistent heuristics. This is noted in (Zhou & Hansen 2002) and in (Nilsson 1998) (p. 153).

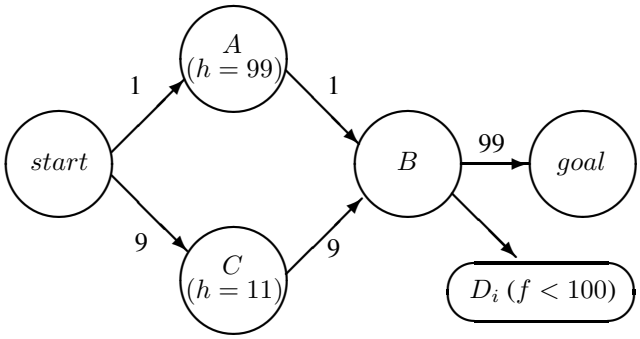

Figure 4: Example where a closed state must be reopened with $pathmax$.

Consider the example in Figure 4 with the admissible, inconsistent heuristic $h(A) = 99$, $h(B) = 0$, and $h(C) = 11$. The optimal path in this example is $start - A - B - goal$, with a cost of 101. A* will expand $start$ and then C (f=20), at which point A and B will be in $OPEN$. A will have $f = 100$ and B, because of $pathmax$, will have $f = 20$ instead of $f = 18$. B will be closed, even though the least-cost path to B (via A) has not been found. A* will then expand the entire set of states D_i before A. At that point A will be expanded, revealing the better path to B, and requiring B and all of the D_i to be expanded for a second time.

$Pathmax$ is half of the two-part strategy introduced by Mero (1984) to exchange heuristic information between a state and its successors when the heuristic is not consistent. Even the full strategy does not guarantee that closed states will not be re-opened, but it can lead to very substantial reductions in the number of node expansions. See (Zhang *et al.* 2009) for full details of Mero's algorithm (called B′), corrections to certain statements in (Mero 1984), and experimental comparisons involving B′.

A* with h=0 and Uniform Costs is not Breadth-first Search

In describing the standard search algorithms, such as Dijkstra's algorithm (Dijkstra 1959) and A*, it is common to give a generic search algorithm and then discuss how it can be specialized for particular circumstances. Often breadth-first search is described as an instance of the generic algorithm, in the special case when all operator costs are the same and $h = 0$ for all states.

This statement is true except for one important detail, the stopping condition. Breadth-first search stops when the goal is first generated (p.47, (Barr & Feigenbaum 1981)). It never needs to put the goal in the $OPEN$ list. A*, Dijkstra's algorithm, and the generic search algorithms in many textbooks, put the goal in the $OPEN$ list and continue searching until it is removed. Breadth-first search can be implemented that way, but it is needlessly inefficient. Consider the example in Figure 5. Expanding $start$ produces an arbitrarily large set of successors, A_i. Expanding any one of these generates $goal$. Breadth-first search would therefore stop after expanding only two states, but A* with $h(s) = 0$ for all states would have to expand all of the A_i before stopping. Note that A* would exactly mimic breadth-first search if it had been given the heuristic $h(goal) = 0$ and $h(s) = 1$ for all other states.

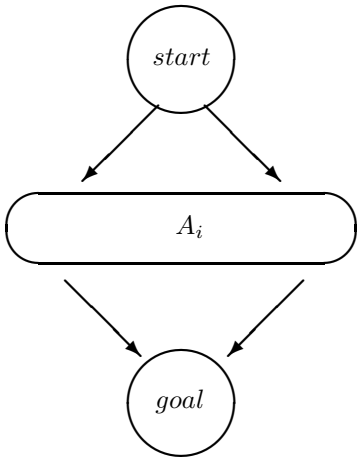

Figure 5: Breadth-first search will stop after expanding two states.

Bidirectional Search Must Continue after the Frontiers Meet In Order to Find Optimal Solutions

Bidirectional search is usually described as stopping when the two search frontiers "intersect", or "meet". This is incorrect. In order to guarantee finding an optimal path bidirectional search must continue to search after a state has been opened in both directions. Bidirectional searches other than bidirectional breadth-first search must even continue after a state has been closed in both directions.

The graph in Figure 6 illustrates why bidirectional breadth-first search cannot stop when the search frontiers first meet. First, $start$ is expanded in the forward direction, putting A and D on the forward-direction $OPEN$ list. Next, $goal$ is expanded in the backwards direction, putting C and E on the backward-direction $OPEN$ list. If A is expanded next in the forward direction and C is expanded next in the backward direction then B will be open in both directions. Search cannot stop at this point and declare the solution to be $start - A - B - C - goal$ because this is not the shortest path.

Although bidirectional breadth-first search cannot stop when a state, B, first becomes open in both directions, the

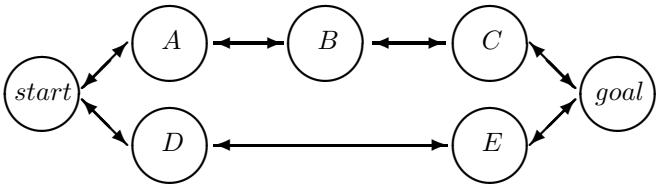

Figure 6: Bidirectional breadth-first search might open B in both directions before finding the shortest path.

path linking $start$ to $goal$ through B cannot be more than one edge longer than the optimal path from $start$ to $goal$. To see this, let us consider the general case, not the specific example in Figure 6. However, to make it easy to relate the general discussion to the example, B will continue to be used to refer to the first state opened in both directions. In general, B is some distance, d_s, from $start$ and some distance, d_g, from $goal$, and therefore the path from $start$ to $goal$ through B is of length $d_s + d_g$. Because B is open in the forward direction the forward-direction $OPEN$ list might contain some states distance $d_s - 1$ from $start$, but it cannot contain states that are closer than that to $start$. Let $OPEN_F(d_s-1)$ denote the set of states open in the forward direction that are distance $d_s - 1$ from $start$. In the example $OPEN_F(d_s - 1) = \{D\}$. Likewise the backward-direction $OPEN$ list might contain some states distance $d_g - 1$ from $goal$, but it cannot contain states that are closer to $goal$. Define $OPEN_B(d_g - 1)$ to be the set of states open in the backward direction that are distance $d_g - 1$ from $goal$. In the example $OPEN_B(d_g - 1) = \{E\}$. $OPEN_F(d_s - 1)$ and $OPEN_B(d_g - 1)$ are the two sets of states that must be examined to determine the shortest path between $start$ and $goal$. The two sets must be disjoint, otherwise B would not be the first state open in both directions. However, there might be an edge connecting a state in $OPEN_F(d_s - 1)$ to a state in $OPEN_B(d_g - 1)$, and if there is, that would constitute a path from $start$ to $goal$ of length $d_s + d_g - 1$, one shorter than the path found through B.

Therefore, what remains to be done once the two search frontiers for bidirectional breadth-first search have met at state B is to search for an edge connecting a state in $OPEN_F(d_s - 1)$ to a state in $OPEN_B(d_g - 1)$. If an edge is found connecting state $D \in OPEN_F(d_s - 1)$ and state $E \in OPEN_B(d_g - 1)$, it is part of the truly shortest path, the remaining portions of the path being the segments from $start$ to D and from $goal$ to E. If no such edge is found then the path through B is optimal.

For bidirectional Dijkstra's algorithm a similar stopping criterion applies (Helgason & Kennington 1993; Nicholson 1966), but for bidirectional A* a different stopping condition is needed (Kwa 1989; Pohl 1969). Every meeting of the two search frontiers constitutes a path from $start$ to $goal$ and therefore provides an upper bound on the optimal path cost. If f_{upper} is the cost of best known path from $start$ to $goal$ bidirectional A* must continue until f_{upper} is less than or equal to the minimum f-value of any state in the $OPEN$ list for either search direction or the sum of the minimum g-values in each $OPEN$ list.[4] When these conditions occur, it will be impossible to find a path costing less than f_{upper} so search can stop and return the path whose cost is f_{upper}.

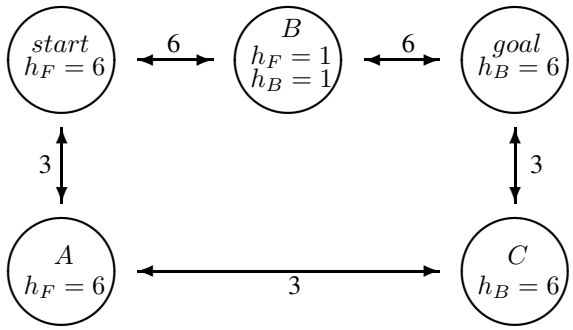

Figure 7: A* will close B in both directions before finding the least-cost path.

Figure 7 gives an example where bidirectional A* will close a state, B, in both directions before finding the optimal path, $start - A - C - goal$. h_F is the heuristic for the forward search, h_B is the heuristic for the backward search. Both are consistent. $start$ is expanded in the forward direction, adding B to the forward-direction $OPEN$ list with $f(B) = 7$ and adding A with $f(A) = 9$. Likewise when $goal$ is expanded B will be added to the backward-direction $OPEN$ list with $f(B) = 7$ and C will be added with $f(C) = 9$. B has the smallest f-value in both $OPEN$ lists and therefore will now be expanded in both directions, finding a path of cost 12 from $start$ to $goal$. Since there are entries on both $OPEN$ lists with $f < 12$ search continues. A is expanded in the forward direction, making contact with the frontier of the backward search at state C. This path has cost 9. Because at least one of the $OPEN$ lists has no state with a smaller f-value search can terminate and return this as the optimal path.

Conclusions

This paper has examined five commonly held beliefs about heuristic search, shown that they are all false, and provided correct versions.

Acknowledgements

This research was supported in part by the Natural Sciences and Engineering Research Council of Canada. Thanks to David Furcy and Rich Korf for numerous suggestions for improvements.

References

Bagchi, A., and Mahanti, A. 1983. Search algorithms under different kinds of heuristics – a comparative study. *Journal of the Association of Computing Machinery* 30(1):1–21.

Barr, A., and Feigenbaum, E. A., eds. 1981. *The Handbook of Artificial Intelligence (Volume 1)*. Addison-Wesley.

[4] The latter condition was brought to my attention by Rich Korf.

Bjornsson, Y.; Enzenberger, M.; Holte, R. C.; and Schaeffer, J. 2005. Fringe Search: Beating A* at pathfinding on game maps. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, 125–132.

Dechter, R., and Pearl, J. 1983. The optimality of A* revisited. In *Proceedings of the National Conference on Artificial Intelligence (AAAI'83)*, 95–99.

Dijkstra, E. W. 1959. A note on two problems in connexion with graphs. *Numerische Mathematik* 1:269–271.

Dinh, H. T.; Russell, A.; and Su, Y. 2007. On the value of good advice: The complexity of A* search with accurate heuristics. In *Proceedings of AAAI*, 1140–1145.

Hart, P.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics* 4:100–107.

Helgason, R. V., and Kennington, J. L. 1993. The one-to-one shortest-path problem: An empirical analysis with the two-tree Dijkstra algorithm. *Computational Optimization and Applications* 1:47–75.

Holte, R. C.; Felner, A.; Newton, J.; Meshulam, R.; and Furcy, D. 2006. Maximizing over multiple pattern databases speeds up heuristic search. *Artificial Intelligence* 170:1123–1136.

Korf, R. E. 1985. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence* 27:97–109.

Kwa, J. 1989. BS*: An admissible bidirectional staged heuristic search algorithm. *Artificial Intelligence* 38(1):95–109.

Manzini, G. 1995. BIDA*: An improved perimeter search algorithm. *Artificial Intelligence* 75(2):347–360.

Martelli, A. 1977. On the complexity of admissible search algorithms. *Artificial Intelligence* 8:1–13.

Mero, L. 1984. A heuristic search algorithm with modifiable estimate. *Artificial Intelligence* 23(1):13–27.

Nicholson, T. 1966. Finding the shortest route between two points in a network. *Computer Journal* 9:275–280.

Nilsson, N. 1998. *Artificial Intelligence: A New Synthesis*. Morgan Kaufman.

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

Pohl, I. 1969. Bi-directional and heuristic search in path problems. Technical Report SLAC-104, Stanford Linear Accelerator Center.

Zhang, Z.; Sturtevant, N.; Holte, R.; Schaeffer, J.; and Felner, A. 2009. A* search with inconsistent heuristics. In *Proceedings of the Twenty-first International Joint Conference on Artificial Intelligence (IJCAI'09)*.

Zhou, R., and Hansen, E. A. 2002. Memory-bounded A* graph search. In *Proceedings of the Fifteenth International FLAIRS Conference (FLAIRS-02)*, 203–209.