Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI 2015)

Speedy versus Greedy Search

Christopher Wilt and **Wheeler Ruml**
Department of Computer Science
University of New Hampshire
Durham, NH 03824 USA
{wilt, ruml} at cs.unh.edu

Abstract

When an optimal solution is not required, satisficing search methods such as greedy best-first search are often used to find solutions quickly. In work on satisficing search, there has been substantial attention devoted to how to solve problems associated with local minima or plateaus in the heuristic function. One technique that has been shown to be quite promising is using an alternative heuristic function that does not estimate cost-to-go, but rather estimates distance-to-go. There is currently little beyond intuition to explain its superiority. We begin by empirically showing that the success of the distance-to-go heuristic appears related to its having smaller local minima. We then discuss a reasonable theoretical model of heuristics and show that, under this model, the expected size of local minima is higher for a cost-to-go heuristic than a distance-to-go heuristic, offering a possible explanation as to why distance-to-go heuristics tend to outperform cost-to-go heuristics.

1 Introduction

Optimal algorithms such as A* [Hart *et al.*, 1968] require impractical amounts of time and/or memory on many problems. For example, memory constraints make it impossible to solve the more difficult instances of the 15-puzzle using the Manhattan distance heuristic and A* without a finely tuned implementation and state-of-the-art hardware. This creates a strong need for non-optimal (satisficing) algorithms that are able to overcome these difficulties. One of the most popular techniques for this is greedy best-first search. Greedy best-first search expands nodes in $h(n)$ order in an attempt to sacrifice solution quality to achieve faster runtime [Doran and Michie, 1966].

Unfortunately, it is rarely the case that it is possible to follow the heuristic directly to a goal, due to local minima and heuristic plateaus. We will say that a node n is in a local minimum if all paths from n to a goal node include at least one node n' such that $h(n') > h(n)$. Informally, this means that in order to get from n to a goal, the h value of the nodes in the path must increase. A local minimum is a maximal connected region of nodes that are all in a local minimum[1]. A heuristic plateau is a maximal connected region of nodes such that all nodes in the region have the same heuristic value.

Because greedy best-first search expands nodes with low h values, both of these phenomena cause problems. In a heuristic plateau, if all nodes have the same h value, greedy search is unable to intelligently differentiate between nodes because the nodes have the same h value. In a local minimum, by hypothesis the nodes with a low h value do not lead to a goal (because they are in the local minimum) and are therefore not desirable nodes to expand if the objective is to solve the problem quickly. In this paper we focus on local minima, because these regions are particularly problematic for greedy best-first search. A heuristic plateau can sometimes be mitigated by tie breaking, but local minima cannot be avoided by greedy best-first search.

Recent work in suboptimal heuristic search and planning has used two kinds of best-first search heuristics: cost-to-go ($h(n)$, "greedy search") [Doran and Michie, 1966], and distance-to-go ($d(n)$, "speedy search") [Ruml and Do, 2007]. $h(n)$ is an approximation of $h^*(n)$, which is the sum of the costs of the edges along a cheapest path starting at n and ending at a goal node. The function $d(n)$ is an approximation of $d^*(n)$, which is the count of edges along a shortest path (measured in count of edges) between n and a goal node.[2]

For minimizing solving time, empirical results strongly favor using a best-first search on $d(n)$ over $h(n)$ [Thayer *et al.*, 2009; Cushing *et al.*, 2010; Richter and Westphal, 2010; Richter *et al.*, 2011]. However, there is currently a lack of understanding of the reasons behind this phenomenon. In this paper, we first show that d is generally more effective for guiding a heuristic search because d tends to have smaller local minima. We also show examples of domains where h has smaller local minima, and how in these domains, greedy best-first search on h is more effective.

Second, we show that, using a random supergraph model of heuristic functions, the expected number of nodes that will be in a local minimum is higher the more the operator costs in the domain vary. This neatly explains the superiority of d, as distance heuristics treat all operators as having the same cost.

[1]In a directed space, these definitions become more complicated.

[2]Some authors define a variant of d that estimates the number of nodes in the cheapest path [Ruml and Do, 2007].

Dom	Cost	Max Local Min Size	Expected Min Size	Exp
Tiles	Unit	392	2.01	801
	Inverse	51,532	87.23	93,010
	Rev Inv	2091	1.94	855
Hanoi	Unit	7,587	1,892.41	36,023
	Rev Sq	35,874	4,415.71	559,250
	Square	2,034	200.82	4,663
TopSpin	Unit	296	250.00	933
	Sum	922	2.65	749
	Stripe	240	2.64	441

Table 1: Sizes of local minima and average expansions required of a greedy best-first search to find a solution.

This work furthers our understanding of suboptimal heuristic search, one of the most scalable planning and problem-solving techniques available.

2 The *d* Heuristic (Usually) Finds Solutions Faster

We begin with a brief overview of the phenomenon we are attempting to explain: the d heuristic employed by speedy search is generally able to outperform the h heuristic employed by greedy best-first search.

We begin our analysis by performing experiments on three benchmark domains, using both unit and non-unit cost functions. For each domain, the problems all have the same underlying graph with the only difference being the edge costs, so a solution to the unit-cost problem is a valid solution to the non-unit problem, and vice-versa. Furthermore, in this analysis, we are only concerned with how long it takes to find a solution, not how much the solution costs, so the edge costs from the underlying problem are only relevant insofar as they are relevant to creating the heuristic.

The first domain we consider is the 3x4 sliding tile puzzle. We used the 3x4 sliding tile puzzle to make it easier to accurately measure the sizes of local minima. The first variant is the standard unit cost function, where h and d are the same, so speedy search and greedy best-first search are the same. We also consider inverse costs, where the price of moving tile n is $\frac{1}{n}$, and reverse inverse, where the price of moving tile n is $\frac{1}{12-n}$. As we can see in Table 1, the unit-cost problems are substantially easier to solve using greedy best-first search as compared to the inverse cost problems, and marginally easier to solve than the reverse inverse problems. This confirms previous results indicating that unit-cost heuristics (d) enable faster search than non-unit (h) ones.

The second domain we consider is the Towers of Hanoi problem with unit costs, but also with square costs (where the cost of moving disk n is n^2) and reverse square costs (where the cost of moving disk n is n^2, but the disks are in reverse order, i.e., the cost of moving disk n out of N is $(N-(n-1))^2$). We considered a set of 51 problems with 12 disks and 4 pegs, and for a heuristic we used a disjoint pattern database, with the first pattern database using the top 4 disks, and the second pattern database using the bottom 8 disks. In this domain, we can see two trends. When we compare greedy best-first search solving the unit-cost problems and the reverse square cost problems, we once again see that the unit-cost problem is easier to solve, as evidenced by it requiring an order of magnitude fewer expansions. If we compare greedy best-first search solving the unit-cost problem and the square cost problem, however, we can see the opposite trend, providing us with our first concrete example of a unit-cost problem that is more difficult to solve than a nonunit-cost problem.

We also consider variants of the popular TopSpin puzzle. We considered 100 problems with 12 disks, and a turnstile that swaps the order of 4 disks. In the unit cost problem, the cost of using the turnstile is 1. With the sum cost function, each disk has an id, and the cost of using the turnstile is the sum of the ids of the disks that are in the turnstile. With the stripe cost function, each disk costs either 1 or 10, depending on whether its id is even or odd, and the cost of using the turnstile is the sum of the costs of the disks that are in the turnstile. We can see in Table 1 that the unit cost problem is once again not the fastest, and in this case, greedy best-first search on the unit-cost problem is slower than greedy best-first search on either kind of non-unit problem.

3 *d* Has Smaller Local Minima

We propose that the expected size of a local minimum using the d heuristic is lower, and that this allows best-first search on d to outperform best-first search on h. There is a clear benefit to greedy best-first search of having small local minima. Unless the initial state is located in a global minimum (a local minimum that contains a goal node), greedy best-first search will begin by expanding all of the nodes in the current local minimum, and will then proceed to look for the goal outside the local minimum.

It is possible to calculate the size of every local minimum in an entire search space by searching backwards from the goal states, expanding nodes in increasing h order. Any node whose h value is less than the highest h value seen thus far is inside a local minimum, since nodes were reverse expanded preferring nodes with low h. The results of this analysis are shown in Table 1. Recall that if the initial state is inside a local minimum, greedy best-first search will expand every single node in the local minimum prior to exiting the local minimum and attempting to find a path to the goal. As we can see in Table 1, as both the expected size of a local minimum and the maximum size of a local minimum increase, the average number of expansions required by a best-first search increases. If we assume the number of local minima encountered by a search is constant, clearly the domain which has larger local minima will be more difficult.

We have just seen that while it is often the case that the unit-cost problems are easier to solve for greedy best-first search, it can also be the case that the unit-cost problems are more difficult to solve, but in either case, greedy best-first search is more effective when the heuristic, whether it is measuring cost or distance, has smaller local minima.

4 Heuristic Gradients

In this section, we describe the general requirements that greedy search places on the gradient induced by the heuristic function, and why that requirement is often better met by heuristics that assume all edges have the same cost.

4.1 High Water Mark Pruning

For every node n, there is a minimum h value, which we denote as h_{hw}, such that at least one path from n to a goal includes no nodes whose h value is greater than h_{hw}. Note that if there are no paths to a goal from n, this value should be infinity. Formally, this quantity is defined as

$$h_{hw}(n) = \min_{paths\ to\ a\ goal} (\max_{p \in path} h(p))$$

In order to find a path to a goal from node n, it is necessary to expand at least one node with an h value of $h_{hw}(n)$ and sufficient to expand all nodes x with $h(x) \leq h_{hw}(n)$.

On problems where there is a solution, greedy search takes advantage of this by never expanding any nodes whose h value is greater than $h_{hw}(root)$. Greedy best-first search terminates when it discovers a path from the start to the goal, and when it terminates, it prunes all nodes on the open list. Because of this, nodes on the open list whose h value is higher than the $h_{hw}(root)$ will never be expanded. As greedy search expands nodes, the minimum h_{hw} of all nodes on the open list either stays the same or decreases, thereby decreasing the maximum h of nodes that will be expanded from that point onwards.

Theorem 1. *On problems for which greedy best-first search terminates, greedy best-first search will expand at least one node with $h(n) = h_{hw}(root)$. Greedy best-first search will not expand any nodes with $h > h_{hw}(root)$.*

Proof. See Wilt and Ruml [2014]. □

The effectiveness of high water mark pruning is driven largely by the relationship between $h_{hw}(n)$ and $h(n)$. For example, suppose $\forall n : h(n) = h_{hw}(n)$. If this is the case, greedy search will be able to expand nodes along a single path leading directly to a goal, assuming optimal tie breaking.

The high water mark is analogous to a dam that the search must flow over. As the difference between $h(root)$ and $h_{hw}(root)$ increases we expect the number of nodes that greedy best-first search will have to expand to simply get over the first heuristic dam (the number of nodes in the local minimum) increases. Thus, it would be beneficial to assess this error in heuristics.

4.2 Heuristic Error

Figure 1 shows the h value of the head of the open list of a greedy best-first search as the search progresses solving a Towers of Hanoi problem with 12 disks, 4 pegs, and a disjoint pattern database, with one part of the disjoint PDB containing 8 disks, and the other containing 4 disks. From this figure, we can see that the h value of the head of the open list of greedy search can fluctuate significantly. These fluctuations can be used to assess inaccuracies in the heuristic function. For example, at about 1,000 expansions the search encounters a node n_{bad} with a heuristic value that is 14, but we can show that the true h value of n_{bad} is at least 20.

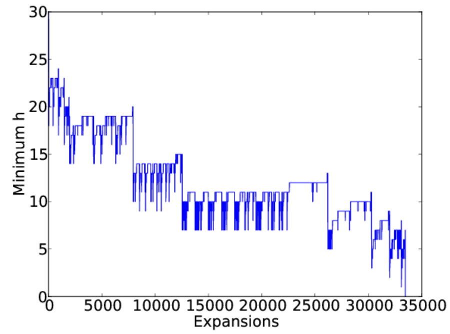

Figure 1: The minimum h value on open as the search progresses, using a disjoint PDB.

After expanding n_{bad}, greedy search then expands a node with an h value of 20 at roughly 7,500 expansions. This allows us to establish that it costs at least 20 to get from n_{bad} to a goal because h is admissible. The general case is expressed as:

Theorem 2. *Consider a node n_{bad} that was expanded by greedy search, and n_{high}, the node with the highest h value that was expanded after n_{bad}, then $h^*(n_{bad}) \geq h(n_{high})$ if h is admissible.*

Proof. See Wilt and Ruml [2014]. □

The genesis of this problem is the fact that n_{bad} is in a local minimum. As discussed earlier, greedy best-first search will expand all nodes in a local minimum in which it expands one node, so clearly larger local minima pose a problem for greedy best-first search.

Heuristic error, defined as deviation from the perfect h^*, is not the root cause of the phenomenon visible in Figure 1. For example, $h(n) = h^*(n) \times 1000$ and $h(n) = h^*(n)/1000$ both have massive heuristic error, but either of these heuristics would be very effective for guiding a best-first search. The problem is the fact that to actually find a goal after expanding n_{bad}, all nodes with $h < h(n_{high})$, and all descendants of those nodes who also have $h < h(n_{high})$, must be cleared from the open list. It is the unproductive expansion of these nodes that causes greedy search to perform poorly.

From the perspective of greedy search, the core of the problem is the difference between $h(n_{high})$ and $h(n_{bad})$, independent of $h^*(n)$. Bringing $h(n_{bad})$ closer to its true value could make it so that n_{bad} is not expanded, but there is another possibility: lowering $h(n_{high})$. This illustrates the importance of the gradient formed by the heuristic when doing greedy best-first search. If the gradient is amenable to following to a goal, greedy best-first search will perform well, but if the gradient is not amenable to following, greedy best-first search will perform poorly.

5 Why *d* is Better than *h*

We now turn to the crucial question raised by these results: why d tends to produce smaller local minima as compared to h, leading it to be a more effective satisficing heuristic.

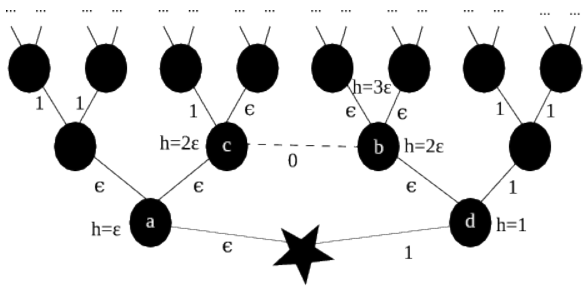

Figure 2: An example of a shortcut tree.

Local Minima are More Likely Using *h*

We begin this analysis by introducing a model of how heuristics are constructed which can be applied to any admissible heuristic. The model was originally created by Gaschnig [1979]. We call this model the shortcut model of heuristic construction.

In any graph g, a node's h^* value is defined as the cost of a cheapest path through the graph from the node to a goal node. In calculating the h value of the node, the shortcut model stipulates that the heuristic constructs a shortest path on a *supergraph* g' which is the same as the original graph, with the exception that additional edges have been added to the graph. The heuristic sometimes includes these edges from the supergraph in its path, which is why it is not always possible to follow the heuristic directly to a goal in the original graph. Any admissible heuristic can be modeled using a supergraph using the degenerate mapping of connecting every node n directly to the goal via an edge with cost $h(n)$. In the context of a pattern database, all nodes that map to the same abstract state are connected to one another by zero cost edges in the supergraph.

Now, we will introduce a special kind of tree which we will use to model heuristic search trees, called a *shortcut tree*, an example of which is shown in Figure 2. A shortcut tree has edge costs assigned uniformly at random from a categorical distribution $opset$ such that the lowest cost edge costs ϵ and the highest cost edge costs 1. Each edge in the shortcut tree is assigned a weight independently from $opset$.

We require that the count of edges in all paths from a leaf to the goal be at least $\frac{1}{\epsilon}$. This means that all paths from a leaf to the root have a cost of at least 1. We model the heuristic of a shortcut tree as a supergraph heuristic that adds edges uniformly at random to the shortcut tree. With some fixed probability $0 \leq p \leq 1$ the supergraph edges have zero cost, but if edges do not have zero cost, they are assigned a cost which is the sum of $n \in [1, N]$ costs drawn from $opset$. A simple example can be seen in Figure 2, where $opset$ only has two possible edge costs, ϵ, and 1. The star represents the goal, which is also the root of the tree.

In Figure 2, all paths from the node b to a goal in the original tree go through node d, but node d has a heuristic value of 1, while node b has a heuristic value of 2ϵ, so node b is inside a local minimum, because going from b to a goal requires at least one node n with $h(n) > h(b)$. The local minimum was caused because node b is connected to node c via a zero cost edge. If node c had a heuristic value greater than 1, the zero cost edge between b and c would not cause a local minimum. Thus, the question of whether or not node b will be in a local minimum is equivalent to asking what the likelihood is that node b is connected to a node whose heuristic value is less than 1.

Shortcut trees have their edge weights and supergraph edges assigned randomly based upon $opset$ and the probability that a supergraph edge is assigned zero cost. As a result, it is impossible to predict exactly what will happen with a particular shortcut tree. It is meaningful, however, to discuss the expected value over all possible assignments of edge weights and supergraph edges. Theorem 3 discusses how the expected probability of a local minimum forming changes as $opset$ changes.

Theorem 3. *Let T be a shortcut tree of fixed height H with edge weight distribution $opset$. As the average value of the items in $opset$ approaches $\frac{1}{H}$, the expected value of the probability that a node whose parent's h value (parent is the neighbor closer to the goal) is at least 1 is inside a local minimum increases. As we increase the prevalence of operators whose cost is not 1, we also increase the expected value of the probability that a node whose parent's h value is at least 1 is inside a local minimum.*

Proof. See Wilt and Ruml [2014]. □

Every node in the tree needs to have an h value that is higher than its parent, otherwise the node will be inside of a local minimum. In particular, nodes whose parents have h values that are higher than 1 that receive h values that are smaller than 1 will be in a local minimum. Theorem 3 shows that two factors contribute to creating local minima in this way: a wide range of operator costs, and an overabundance of low cost operators. Both of these factors make sense. When the cheap edges are relatively less expensive, there are going to be more nodes in the tree whose cost is smaller than 1. This increases the likelihood that a node that needs a high heuristic value is connected in the supergraph to a node with a low heuristic value because there are more nodes with low heuristic values. Likewise, when the prevalence of low cost edges increases, there are more parts of the tree with deceptively low heuristic values that look promising for a best-first search to explore.

To the extent that shortcut trees model a given heuristic, Theorem 3 offers an explanation of why guiding a best-first search with d is likely to be faster than guiding a best-first search with h. With d, the heuristic pretends that $opset$ only contains the value 1. Thus, as we morph d into h by lowering the average value in $opset$, and increasing the prevalence of operators whose cost is not 1 we increase the probability that low h nodes are inside a local minimum.

Theorem 3 also tells us that when doing best-first search, one possible source of inefficiency is the presence of many low cost edges either in the original graph or the supergraph, because these edges cause local minima. Low cost edges increase the probability that the h is computed from a supergraph path that bypasses a high h region, causing a local minimum, which best-first search on h will have to fill in.

One limitation of the analysis of Theorem 3 is that it considers only trees, while most problems are better represented by graphs. Fortunately, the analysis done in Theorem 3 is also relevant to graphs. The difference between search in a graph and a tree is the fact that there are no duplicate nodes in a tree. These duplicate nodes can be modeled using zero cost edges between the nodes in the tree that represent the same node in the graph. This makes it so that there are two kinds of zero cost edges: ones that were added because the problem is a graph, and zero cost edges from the supergraph. If we assume that the zero cost edges that convert the tree to a graph are also uniformly and randomly distributed throughout the space just like the zero cost edges from the supergraph, we arrive at precisely the same conclusion from Theorem 3.

If we consider a supergraph heuristic for an arbitrary graph, the edges involved in h^* form a tree, as long as we break ties. The complication with this approach is the fact that if a node has a high h value (say greater than 1), it may be possible to construct a path that bypasses the node in the graph, something that is not possible in a tree. This can cause problems with Theorem 3 because a single high h node is not enough to cause a local minimum – one needs a surrounding "dam" on all sides of the minimum. In this case, we can generalize Theorem 3 by specifying that the high h node is not simply a single node, but rather a collection of nodes that all have high h with the additional restriction that one of the nodes must be included in any path to the goal.

Theorem 3 assumes the edge costs and shortcut edges are uniformly distributed throughout the space, but the edge costs and shortcut edges may not be uniformly distributed throughout the space. If we do not know anything about a particular heuristic, applying Theorem 3, which discusses the expected properties of a random distribution of edge costs and supergraph edges, may be the best we can do. To the extent that the shortcut model is relevant, it suggests that h has more local minima.

5.1 Local Minima Can be Costly for Greedy Best-First Search

In the previous section, we saw that local minima were more likely to form when the difference in size between the large and small operators increased dramatically. We also saw that, as the low cost operators increased in prevalence, local minima also became more likely to form. In this section we address the consequences of the local minima, and how those consequences are exacerbated by increasing the size difference between the large and small operators and the increased prevalence of low cost operators.

We begin by assuming that the change in h between two adjacent nodes n_1 and n_2 is often bounded by the cost of the edge between n_1 and n_2.[3]

Consider the tree in Figure 3. In this tree, we have to expand all of the nodes whose heuristic value is less than 1, because the only goal in the space is a descendant of a node whose h value is 1. The core of the problem is the fact that node N was assigned a heuristic value that is way too low. If we restrict the change in h to be smaller than the operator cost, in order to go from 1 to ϵ the operator must have a cost of at least $1-\epsilon$. If the operator's cost is less than $1-\epsilon$, the heuristic on N would have to be higher than ϵ. The tree rooted at N continues infinitely, but if h increases by ϵ at each transition, it will take $1/\epsilon$ transitions before h is at least 1. This means the subtree contains $2^{1/\epsilon}$ nodes, all of which would be expanded by a greedy best-first search. The tree in Figure 3 represents the best possible outcome for greedy best-first search, where the heuristic climbs back up from an error as fast as it can. In a more antagonistic case, h could either fall or stay the same, which would exacerbate the problem, adding even more nodes to the local minimum.

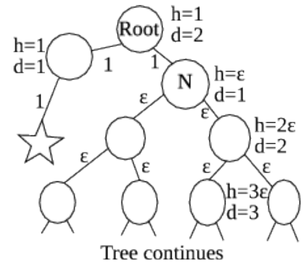

Figure 3: A search tree with a local minimum

If we substitute d for h, the ϵ edges change to cost 1, which makes it so the subtree expanded by greedy best-first search only contains 1 node. The number of nodes that can fit in a local minimum caused by a single error is much larger if the low cost edges in the graph have very low cost. The idea here is very similar to Corollary 1 of Wilt and Ruml [2011]'s analysis of A* except in this case, g is not contributing to escaping the local minimum, because greedy best-first search does not consider g when evaluating nodes. In this way, we see how local minima can be much more severe for h than for d, further explaining the superiority of d.

5.2 Summary

Theorem 3 discusses how likely a local minimum is to form, and shows that as we increase the prevalence of low cost edges or decrease the cost of the low cost edges, the likelihood of creating a local minimum increases. The local minima created have high water marks that are determined by the high cost edges in the graph. We then showed that if we have a local minimum whose height (the difference in h between the node in the local minimum with the lowest h of all nodes in the local minimum, and the lowest h value of all nodes that lead out of the local minimum) is the same as the high cost edge, the number of nodes that can fit inside of the local minimum can be exponential in the ratio of the high cost edge to the low cost edge, demonstrating that the performance penalty associated with even a single error in the heuristic is very severe, and grows exponentially as the low cost edges decrease in cost. Using the d heuristic instead of h mitigates these problems, because there are no high cost edges or low cost edges.

While it is generally true that the d heuristic is more useful than h, note that some heuristics do not follow this general trend. For example, best-first search using the h heuristic on the Towers of Hanoi using the square cost function is

[3] Wilt and Ruml [2011] showed that this is a reasonable restriction, and that many heuristics obey this property.

faster than the d heuristic. The reason behind this trend is the fact that Theorem 3 only discusses the expected value across all possible heuristics that add the same number of zero cost edges to the graph. Which zero cost edges get added clearly has a major effect on how well a particular heuristic will work.

6 Related Work

A number of algorithms make use of a distance-based heuristic. For example, Explicit Estimation Search [Thayer and Ruml, 2011] uses a distance-based heuristic to try and find a goal quickly. Deadline Aware Search [Dionne *et al.*, 2011] is an algorithm that uses distance estimates to help find a solution within a specified deadline. The LAMA 2011 planner [Richter *et al.*, 2011; Richter and Westphal, 2010], winner of the 2011 International Planning Competition, uses a distance-based heuristic to form its first plan.

Chenoweth and Davis [1991] discuss a way to bring A* within polynomial runtime by multiplying the heuristic by a constant. With a greedy best-first search, the constant is effectively infinite, because we completely ignore g. One limitation of this analysis is that it leaves open the question of what h should measure. Moreover, it is unclear from their analysis if it is possible to put too much weight on h, which is what a best-first search on either d or h does.

Cushing, Benton, and Kambhampati [2010; 2011] argue that cost-based search (using h) is harmful because search that is based on cost is not interruptible. They argue that the superiority of distance-based search stems from the fact that the distance-based searches can provide solutions sooner, which is critical if cost-based search cannot solve the problem, or requires too much time or memory to do so. This work, however, does not directly address the more fundamental question of when cost-based search is harmful, and more importantly, when cost-based search is helpful.

Wilt and Ruml [2011] demonstrated that when doing best-first search with a wide variety of operator costs, the penalty for a heuristic error can introduce an exponential number of nodes into the search. They then proceed to show that this exponential blowup can cause problems for algorithms that use h exclusively, rendering the algorithms unable to find a solution. Last, they show that algorithms that use d are still able to find solutions. This work shows the general utility of d, but leaves open the question of precisely why the algorithms that use d are able to perform so well.

Another approach to enhancing greedy best-first search is to ignore the heuristic under certain circumstances. On certain domains, this technique can improves performance and coverage, increasing the total number of problems that can be solved.

Hoffmann discusses how the topology induced on the search space by the heuristic affects the enforced hill climbing algorithm, proving that for many STRIPS planning benchmarks, the algorithm will exhibit polynomial runtime [Hoffmann, 2011; 2005].

7 Conclusion

It is well known that searching on distance can be faster than searching on cost. We provide evidence that suggests that the root cause of this is the fact that the d heuristic tends to produce smaller local minima compared to the h heuristic. We also saw that greedy best-first search on h can outperform greedy best-first search on d if h has smaller local minima than d.

This naturally leads to the question as to why the d heuristic tends to have smaller local minima as compared to the h heuristic. We showed that if we model the search space using a tree and use a random supergraph heuristic, we expect that the d heuristic will have smaller local minima compared to the h heuristic, which explains why researchers have observed that searching on d tends to be faster than searching on h. Given the ubiquity of large state spaces and tight deadlines, we hope that this work spurs further investigation into the behavior of suboptimal search algorithms.

8 Acknowledgements

We gratefully acknowledge support from the NSF (grant 1150068). This paper is a shorter presentation of a SoCS 2014 paper [Wilt and Ruml, 2014].

References

[Chenoweth and Davis, 1991] Stephen V. Chenoweth and Henry W. Davis. High-performance A* search using rapidly growing heuristics. In *Proceedings of the Twelfth International Joint Conference on Articial Intelligence*, pages 198–203, 1991.

[Cushing *et al.*, 2010] William Cushing, J. Benton, and Subbarao Kambhampati. Cost based search considered harmful. In *Proceedings of the Third Symposium on Combinatorial Search*, 2010.

[Cushing *et al.*, 2011] William Cushing, J. Benton, and Subbarao Kambhampati. Cost based search considered harmful. http://arxiv.org/abs/1103.3687, June 2011.

[Dionne *et al.*, 2011] Austin Dionne, Jordan T. Thayer, and Wheeler Ruml. Deadline-aware search using on-line measures of behavior. In *Proceedings of the Fourth Annual Symposium on Combinatorial Search*, 2011.

[Doran and Michie, 1966] J. E. Doran and D. Michie. Experiments with the graph traverser program. In *Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences*, pages 235–259, 1966.

[Gaschnig, 1979] John Gaschnig. A problem similarity approach to devising heuristics: First results. In *Proceedings of the Sixth International Joint Conference on Articial Intelligence (IJCAI-79)*, 1979.

[Hart *et al.*, 1968] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, SSC-4(2):100–107, July 1968.

[Hoffmann, 2005] Jörg Hoffmann. Where 'ignoring delete lists' works: Local search topology in planning benchmarks. *Journal of Artifial Intelligence Research*, 24:685–758, 2005.

[Hoffmann, 2011] Jörg Hoffmann. Analyzing search topology without running any search: on the connection between causal graphs and h+. *J. Artif. Int. Res.*, 41(2):155–229, May 2011.

[Richter and Westphal, 2010] Silvia Richter and Matthias Westphal. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artifial Intelligence Research*, 39:127–177, 2010.

[Richter *et al.*, 2011] Silvia Richter, Matthias Westphal, and Malte Helmert. LAMA 2008 and 2011. In *International Planning Competition 2011 Deterministic Track*, pages 117–124, 2011.

[Ruml and Do, 2007] Wheeler Ruml and Minh B. Do. Best-first utility-guided search. In *Proceedings of the Twenty Second h International Joint Conference on Articial Intelligence (IJCAI-07)*, pages 2378–2384, 2007.

[Thayer and Ruml, 2011] Jordan Tyler Thayer and Wheeler Ruml. Bounded suboptimal search: A direct approach using inadmissible estimates. In *Proceedings of the Twenty Sixth International Joint Conference on Articial Intelligence (IJCAI-11)*, pages 674–679, 2011.

[Thayer *et al.*, 2009] Jordan T. Thayer, Wheeler Ruml, and Jeff Kreis. Using distance estimates in heuristic search: A re-evaluation. In *Proceedings of the Second Symposium on Combinatorial Search*, 2009.

[Wilt and Ruml, 2011] Christopher Wilt and Wheeler Ruml. Cost-based heuristic search is sensitive to the ratio of operator costs. In *Proceedings of the Fourth Symposium on Combinatorial Search*, July 2011.

[Wilt and Ruml, 2014] Christopher Wilt and Wheeler Ruml. Speedy versus greedy search. In *Proceedings of the Seventh Symposium on Combinatorial Search*, July 2014.