When does Weighted A* Fail?

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

Abstract

Weighted A* is the most popular satisficing algorithm for heuristic search. Although there is no formal guarantee that increasing the weight on the heuristic cost-to-go estimate will decrease search time, it is commonly assumed that increasing the weight leads to faster searches, and that greedy search will provide the fastest search of all. As we show, however, in some domains, increasing the weight slows down the search. This has an important consequence on the scaling behavior of Weighted A*: increasing the weight ad infinitum will only speed up the search if greedy search is effective. We examine several plausible hypotheses as to why greedy search would sometimes expand more nodes than A* and show that each of the simple explanations has flaws. Our contribution is to show that greedy search is fast if and only if there is a strong correlation between $h(n)$ and $d^*(n)$, the true distance-to-go, or if the heuristic is extremely accurate.

Introduction

Many of the most effective algorithms in satisficing search have a weight parameter that can be used to govern the balance between solution quality and search effort. The most famous of these is Weighted A* (Pohl 1970), which expands nodes in f' order, where $f'(n) = g(n) + w \cdot h(n) : w \in (1, \infty)$. Weighted A* is used in a wide variety of applications. For example, the Fast Downward planner (Helmert 2006) uses Weighted A*, and LAMA (Richter and Westphal 2010) also uses a variant of Weighted A*. Weighted A* is also used extensively for planning for robots (Likhachev, Gordon, and Thrun 2003).

In addition to that, Weighted A* is a component of a number of anytime algorithms. For example, Anytime Restarting Weighted A* (Richter, Thayer, and Ruml 2009) and Anytime Repairing A* (Likhachev, Gordon, and Thrun 2003) both use Weighted A*. Anytime Nonparametric A* (van den Berg et al. 2011) doesn't directly use Weighted A* like the preceding anytime algorithms, but this algorithm requires greedy search (Doran and Michie 1966) to find a solution quickly. All of these anytime algorithms have, built in, the implicit assumption that Weighted A* with a high weight or greedy search will find a solution faster than A* or Weighted A* with a small weight.

In many popular heuristic search benchmark domains (sliding tile puzzles, grid path planning, Towers of Hanoi, TopSpin, robot motion planning, traveling salesman problem, for example) increasing the weight does lead to a faster search, until the weight becomes so large that Weighted A* has the same expansion order as greedy search, which results in the fastest search. As we shall see, however, in some domains greedy search performs worse than Weighted A*, and is sometimes even worse than A*.

We examine several reasonable hypotheses as to why this might occur, and show that each has flaws. The first hypothesis we consider is the idea that when the percentage error of the heuristic is high, greedy search performs poorly, and conversely when the percent error in the heuristic is low, greedy search performs well. Next we consider the size of the local minima in the domain, hypothesizing that greedy search is effective in domains with small local minima. We then consider the strength of the correlation between $h(n)$ (the estimated cost of getting to a goal from node n) and $h^*(n)$ (the true cost of getting to a goal from node n), and attempt to classify domains as either greedy search friendly or greedy search unfriendly based upon this correlation. Finally, we consider the strength of the correlation between $d^*(n)$ (the number of nodes between n and the nearest goal, measured in edge count) and $h(n)$, and show that this number is the most important for predicting when greedy search will perform poorly, unless the correlation between $h(n)$ and $h^*(n)$ is nearly perfect. This conclusion is based on empirical analysis in a variety of benchmark domains.

We show that the failure of greedy search is not merely a mathematical curiosity, only occurring in hand crafted counterexamples, but rather a phenomenon that can occur in real domains. Our contribution is to identify conditions when this occurs, knowledge which is important for anyone using a suboptimal search. This is also an important first step in a predictive theoretical understanding of the behavior of suboptimal heuristic search.

Previous Work

Gaschnig (1977) describes how to predict the worst case number of nodes expanded by A*, and also discusses how weighting the heuristic can affect the worst case final node expansion count. His predictions, however, have two limi-

tations. First, the predictions assume the search space is a tree, and not a graph, as is the case for many applications of heuristic search. In addition to that, the worst case predictions only depend the amount of error present in the heuristic, where error is measured as deviation from $h^*(n)$. As we shall see, looking only at raw error leads to incorrect conclusions.

Chenoweth and Davis (1991) show that if the heuristic is 'rapidly growing with logarithmic cluster', the search can be done in polynomial time. A heuristic is rapidly growing with logarithmic cluster if, for every node n, $h(n)$ is within a logarithmic factor of a monotonic function f of $h^*(n)$, and f grows at least as fast as the function $g(x) = x$. On a finite graph, any heuristic is rapidly growing with logarithmic cluster. The claims presented by Chenoweth and Davis (1991) are significant when $h^*(n)$ can grow arbitrarily large, which can only occur when the graph is infinite. For all benchmark domains we consider, this is not the case.

Hoffmann (2005) discusses the performance of the Fast Forward heuristic (Hoffmann and Nebel 2001). They show that the Fast Forward heuristic can be proved to be extremely effective for many planning benchmark domains when used with a variant of greedy hill-climbing, allowing many of the planning benchmark domains to be solved quickly, sometimes in linear time.

A number of works consider the question of predicting search algorithm performance (Korf, Reid, and Edelkamp 2001; Pearl 1984; Helmert and Röger 2007), although the subject attracting by far the most attention is determining how many nodes will be expanded by an optimal search algorithm, which does not lend any insight into the question of whether or not greedy search is likely to perform well or poorly. Lelis, Zilles, and Holte (2011) did empirical analysis of suboptimal search algorithms, predicting the number of nodes that would be expanded by Weighted IDA*, but it remains an open question determining whether or not the Weighted IDA* predictions can tell us if increasing the weight too far can be detrimental for that algorithm.

Korf (1993) provides an early discussion of how increasing the weight may actually be bad, showing that when recursive best first search or iterative deepening A* is used with a weight that is too large, expansions actually increase. This paper is also an early example of exploring how the weight interacts with the expansion count, something central to this paper.

When is increasing w bad?

In order to get a better grasp on the question of when increasing the weight is bad, we first need some empirical data, examples of when increasing the weight is either good, speeding up search, or bad, slowing down search. We consider six standard benchmark domains: the sliding tile puzzle, the Towers of Hanoi puzzle, grid path planning, the pancake problem, TopSpin, and dynamic robot navigation. Since we need to know the optimal solution, we are forced to use somewhat smaller puzzles than it is possible to solve using state of the art suboptimal searches. Our requirement for problem size was that the problem be solvable by A*, Weighted A*, and greedy search in main memory (eight gigabytes). We selected these domains because they represent a wide variety of interesting heuristic search features, such as branching factor, state space size, and solution length. Basic statistics about each of these domain variants are summarized in Table 1.

Domain	Solution Length	Total States	Branching Factor
Dynamic Robot	187.45	20,480,000	0-240
Hanoi (14)	86.92	268,435,456	6
Pancake (40)	38.56	8×10^{47}	40
11 Tiles (unit)	36.03	239,500,800	1-3
Grid	2927.40	1,560,000	0-3
TopSpin (3)	8.52	479,001,600	12
TopSpin (4)	10.04	479,001,600	12
11 Tiles (inverse)	37.95	239,500,800	1-3
City Navigation 3 3	15.62	22,500	3-8
City Navigation 4 4	14.38	22,500	3-10
City Navigation 5 5	13.99	22,500	3-12

Table 1: Domain Attributes for benchmark domains considered

For the 11 puzzle (3x4), we used random instances and the Manhattan distance heuristic. We used the 11 puzzle, rather than the 15 puzzle, because we also consider the sliding tile puzzle with non-unit cost functions, which are not practical to solve optimally for larger puzzles. In addition to that, A* runs out of memory solving a 15 puzzle for the more difficult instances. The non-unit version of sliding tile puzzle we consider has the cost of moving a tile n as $1/n$. The Manhattan distance heuristic, when weighted appropriately, is both admissible and consistent. For the Towers of Hanoi, we considered the 14 disk 4 peg problem, and used two disjoint pattern databases, one for the bottom 12 disks, and one for the top two disks (Korf and Felner 2002). For the pancake problem, we used the gap heuristic (Helmert 2010). For grid path planning, we used maps that were 2000x1200 cells, with 35% of the cells blocked, using the Manhattan distance heuristic with four way movement. For the TopSpin puzzle, we considered a problem with 12 disks with a turnstile that would turn either three or four disks, denoted by TopSpin(3) or TopSpin(4) to differentiate between the two varieties. For a heuristic, we used a pattern database with 6 contiguous disks present, and the remaining 6 disks abstracted. For the dynamic robot navigation problem, we used a 8000x8000 world, with 32 headings and 16 speeds. Space was discretized to produce cells that were 40x40, so the world contained 200x200 cells.

We also introduce a new domain we call City Navigation, designed to simulate navigation using a system similar to American interstate highways or air transportation networks. In this domain, there are cities scattered randomly on a 100x100 square, connected by a random tour to ensure solvability. Each city is also connected to its n_c nearest neighbors. All links between cities cost the Euclidean distance + 2. Each city contains a collection of locations, randomly scattered throughout the city (which is a 1x1 square). Locations in a city are also connected in a random tour, with

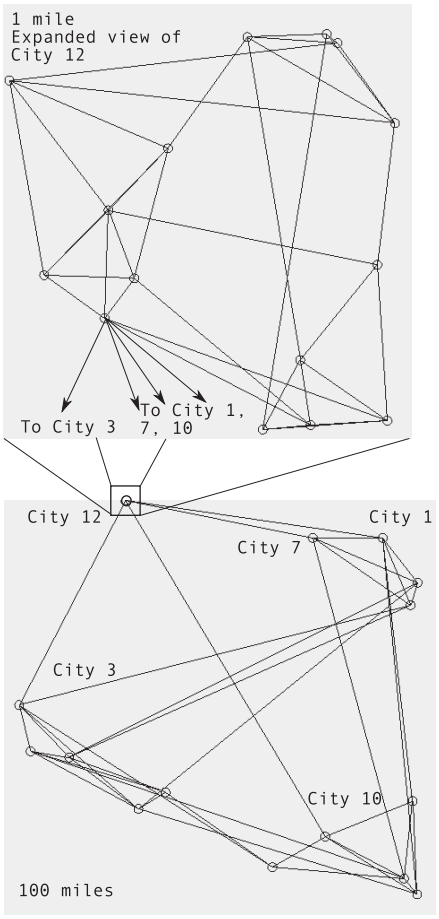

Figure 1: A city navigation problem with $n_p = n_c = 3$, with 15 cities and 15 locations in each city.

each place connected to the nearest n_p places. Links between places cost the true distance multiplied by a random number between 1 and 1.1. Within each city is a special nexus node that contains all connections in and out of this city. The goal is to navigate from a randomly selected start location to a randomly selected end location. For example, we might want to go from Location 3 in City 4 to Location 23 in City 1. Each city's nexus node is Location 0, so the goal for the example problem is to navigate from Location 3 to Location 0 in City 4, then find a path from City 4 to City 1, then a path from Location 0 in City 1 to Location 23 in City 1. In this domain, solutions vary in length, and it is straightforward to manipulate the accuracy of the heuristic.

An example instance of this type can be seen in Figure 1. The circles in the top part of the figure are locations, connected to other locations. The nexus node, Location 0, is also connected to the nexus nodes of neighboring cities. The bottom part of the figure shows the entire world, with cities shrunk down to a circle. City Navigation instances are classified by n_c and n_p, so we call a particular kind of City Navigation problem City Navigation n_p n_c. We consider problems with varying numbers of connections, but always having 150 cities and 150 places in each city. Since each location within a city has a global position, the heuristic is direct Euclidean distance.

Figure 2 and 3 show the number of expansions required by A*, greedy search, and Weighted A* with weights of 1.1, 1.2, 2.5, 5, 10, and 20. These plots allow us to compare greedy search with Weighted A* and A*, and to determine whether increasing the weight speeds up the search, or slows down the search.

Looking at the plots in Figure 2, it is easy to see that as we increase the weight the number of expansions goes down. In Figure 3, the opposite is true. In each of these domains, increasing the weight initially speeds up the search, as A* is relaxed into Weighted A*, but as Weighted A* transforms into greedy search, the number of nodes required to solve the problem increases. In two of the domains, TopSpin with a turnstile of size 4 and City Navigation 3 3, the number of nodes expanded by greedy search is higher than the number of nodes expanded by A*.

What makes increasing w bad?

We have established that increasing the weight in Weighted A* does not always speed up the search. Next, we consider a number of hypotheses as to why this would occur.

Hypothesis 1 *Greedy search performs poorly when the percent error in $h(n)$ is high.*

The percent error in the heuristic is defined as $\frac{h^*(n)-h(n)}{h^*(n)}$. Since greedy search increases the importance of the heuristic, it is reasonable to conclude that if the heuristic has a large amount of error, relying upon it heavily, as greedy search does, is not going to lead to a fast search.

In Table 2, we have the average percent error in the heuristic for each of the domains considered. Surprisingly, the average percentage error bears little relation to whether or not greedy search will be a poor choice. Towers of Hanoi, unit tiles, and TopSpin(3), three domains where greedy search is effective, have as much or more heuristic error than domains where greedy search works poorly. This leads us to conclude that you cannot measure the average heuristic percent error and use this to determine whether or not increasing the weight will speed up or slow down search.

Intuitively, this makes sense, as greedy search only really requires that the nodes get put in $h^*(n)$ order by the heuristic. The exact size, and therefore error, of the heuristic is unimportant, but size has a huge effect on the average percent error. This can be seen if we consider the heuristic $h(n) = \frac{h^*(n)}{N}, N \in \mathbb{N}$, which will always guide greedy search directly to an optimal goal, while exhibiting arbitrarily high average percent error in the heuristic as N increases.

Hypothesis 2 *Greedy search performs poorly when local minima in $h(n)$ contain many nodes.*

The concept of a local minimum in $h(n)$ is not rigorously defined, so the first step is to precisely define what a local minimum is. The first, and simplest definition, is to define a local minimum as a maximal set of connected nodes N such that for each node $n \in N$, every path from n to a node $n' \notin N$ includes at least one node n_{max} such that $h(n) < h(n_{max})$.

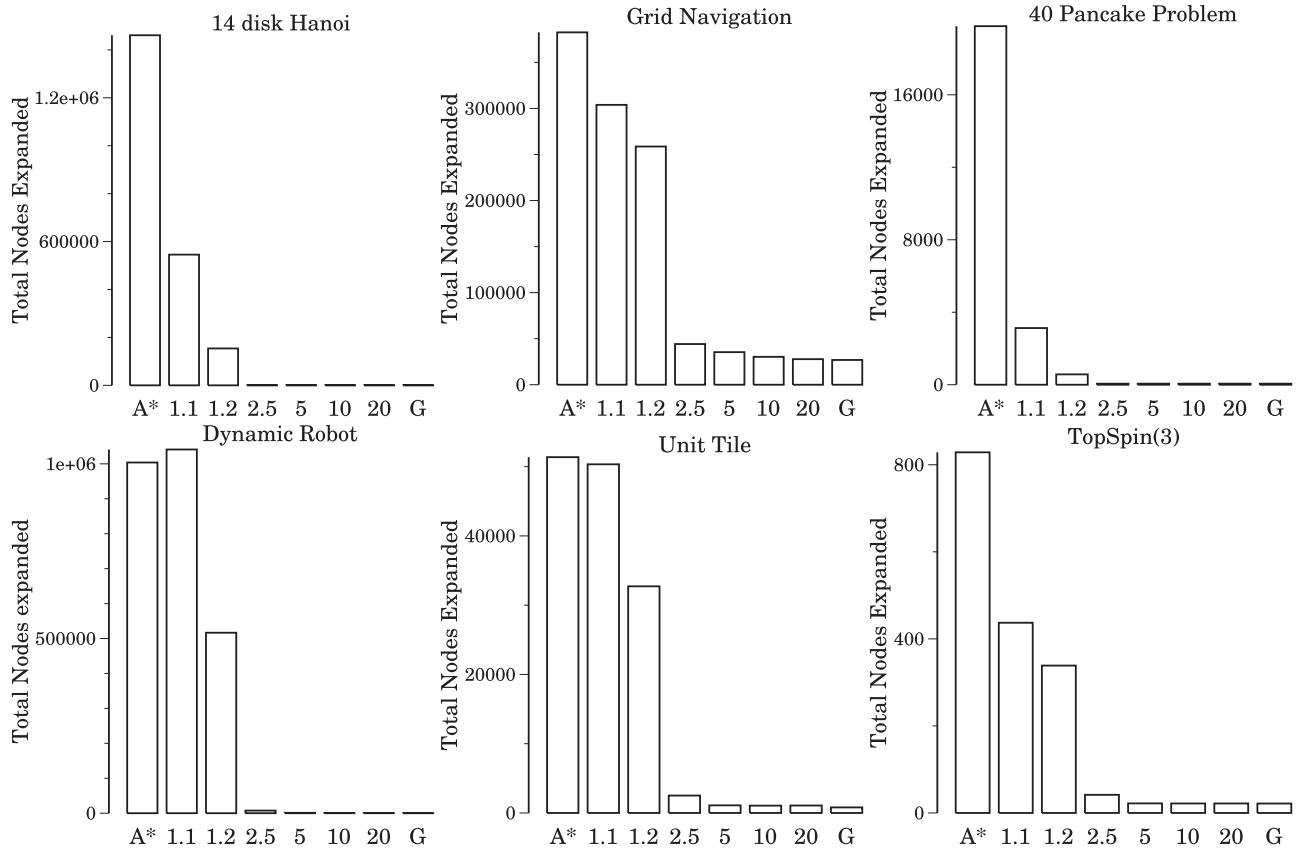

Figure 2: Domains where increasing the weight speeds up search

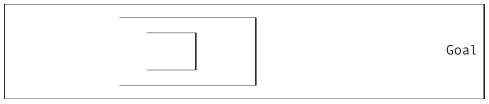

Figure 4: Picture of one or two of local minima, depending on how they are defined

The "size" of a local minimum under this metric has little bearing on how effective greedy search is. For example, in the sliding tile domain with unit cost, an average local minimum contains 2.48 nodes, and in the sliding tile puzzle with inverse costs, where greedy search performs very poorly, an average local minimum contains 2.47 nodes. Thus, the size of a local minimum measured in this way cannot be used to predict whether or not greedy search is going to perform poorly. The reason is that local minima can be nested: once the search escapes the first local minimum, it is possible that it is inside another local minimum.

The solution, then, is to change the definition of a local minimum to verify that the path out of the first local minimum actually terminates in a goal node, as opposed to possibly leading into another local minimum. One modified definition of a local minimum is a maximal set of connected nodes N such that for each node $n \in N$, every path from n to a goal node includes at least one node n_{max} such that $h(n) < h(n_{max})$. The difference between this definition, and the previous one, is what happens to the small cup inside the large cup shown in Figure 4. Under the first definition, the small inner cup is its own local minimum, but under the second definition, the small inner cup is not a local minimum on its own.

Although this improved definition solves the problem of nested local minima, it still does not help us determine whether or not greedy search will work well. The bottom picture in Figure 5 shows an example where a large local minimum can be solved easily by greedy search. In this graph, there is a single local minimum which can be made to be arbitrarily large. This region is a local minimum under either definition we have thus far considered. In this domain, however, greedy search will expand only one extra node in the worst case. If run on this domain, greedy search will follow the heuristic until it expands the left $h(n) = 1$ node. In the worst case, it will then expand all unexpanded nodes with $h(n) = 2$, which can introduce at worst one unnecessary expansion. At this point, the right node with $h(n) = 1$ is guaranteed to have been discovered, and expanding it leads to the goal. In this domain, despite the large size of the local minimum, greedy search exhibits linear run time.

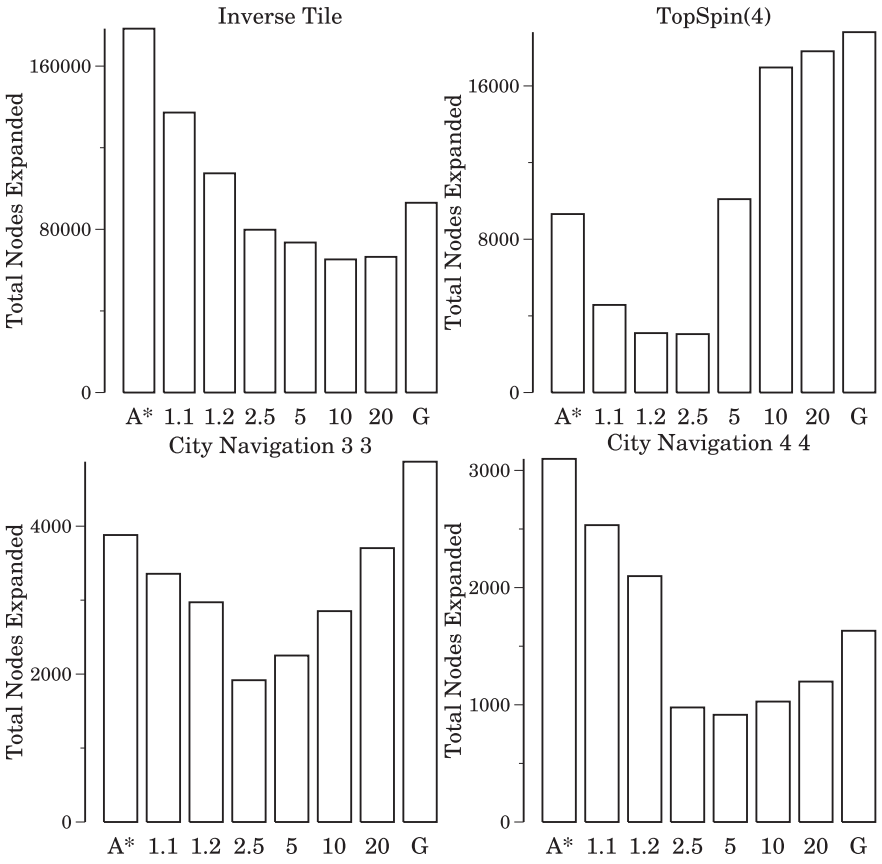

Figure 3: Domains where increasing the weight slows down search

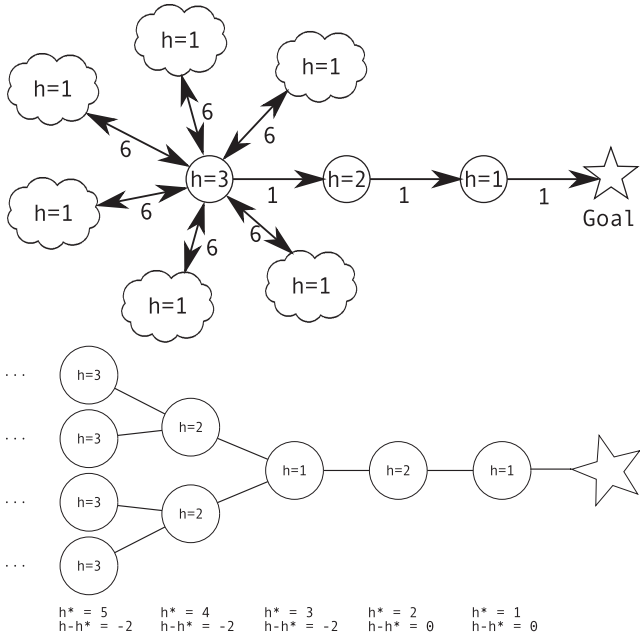

Figure 5: Graphs where local minimum size doesn't matter

The top graph in 5 shows an example where greedy search will perform poorly even if individual local minima are small. In this graph, all edge costs are labeled, and edges can be traversed in the directions of the arrows. The clouds represent individual local minima where all the nodes in the cloud have $h(n) = 1$. If run on this graph from a start state in one of the local minima, greedy search will eventually expand the $h(n) = 3$ node, which will generate one node in each local minima. The node with $h(n) = 2$ will not be expanded until every single node in all of the local minima has been expanded.

In this situation, the size of the individual local minima has no bearing on how many nodes are expanded by greedy search, because the number of local minima present in the domain can be made arbitrarily large independent of the size of the local minima. A* may expand all of the local minimum containing the initial state, but will not expand any nodes in the other local minima because those nodes will have high $f(n)$ values due to having a high $g(n)$ value.

We have shown that if local minima are large, greedy search may nonetheless perform well, and that even if local minima are small, greedy search may perform poorly. This leads us to conclude that the size of the local minima cannot be used to predict whether or not greedy search will be effective.

	Domain	Heuristic % Error	$h(n)$-$h^*(n)$ Correlation (Pearson)	$h(n)$-$h^*(n)$ Correlation (Spearman)	$h(n)$-$d^*(n)$ Correlation (Pearson)	$h(n)$-$d^*(n)$ Correlation (Spearman)
Greedy Works	Towers of Hanoi	29.47	0.9652	0.9434	0.9652	0.9434
	Grid	12.78	0.9790	0.9704	0.9790	0.9704
	Pancake	2.41	0.9621	0.9607	0.9621	0.9607
	Dynamic Robot	15.66	0.9998	0.9983	0.9989	0.9968
	Unit Tiles	33.37	0.7064	0.7090	0.7064	0.7090
	TopSpin(3)	20.94	0.5855	0.5211	0.5855	0.5211
Greedy Fails	TopSpin(4)	20.25	0.2827	0.3924	0.2827	0.3924
	Inverse Tiles	29.49	0.6722	0.6584	0.3670	0.3375
	City Nav 3 3	44.51	0.5688	0.6132	0.0246	-0.0255
	City Nav 4 4	37.41	0.7077	0.7518	0.0853	0.1641

Table 2: Average % error, correlation between $h(n)$ and $h^*(n)$, and correlation between $h(n)$ and $d^*(n)$ in different domains

Domain	Heuristic % Error	$h(n)$-$h^*(n)$ Correlation (Pearson)	$h(n)$-$h^*(n)$ Correlation (Spearman)	$h(n)$-$d^*(n)$ Correlation (Pearson)	$h(n)$-$d^*(n)$ Correlation (Spearman)
City Nav 5 5	31.19	0.9533	0.9466	0.0933	0.0718

Table 3: Average % error, correlation between $h(n)$ and $h^*(n)$, and correlation between $h(n)$ and $d^*(n)$ in City Nav 5 5

Hypothesis 3 *Greedy search performs poorly when local minima in the $h(n) - h^*(n)$ function contain many nodes.*

Another hypothesis we consider, presented by Likhachev (2005), is that when the local minima in the function $h(n) - h^*(n)$ are small (with size measured in nodes), search will be fast, but if the local minima are large, search will be slow. Intuitively, $h(n) - h^*(n)$ measures the amount of error in the heuristic.

A node n is in a local minimum of the $h(n) - h^*(n)$ function if $h(n) \neq h^*(n)$. A local minimum is a maximal collection of connected nodes that are in a local minimum. As an example, consider the top part of Figure 5. In this figure, the nodes represented by circles all have $h(n) = h^*(n)$, and are therefore not part of a local minimum. The nodes in the clouds all have heuristic error, so these nodes are in a local minimum. Each cloud is a single local minimum, because none of the clouds are connected to another cloud via a path containing only nodes in a local minimum, as any path from one cloud to another goes through the $h(n) = 3$ node, which is not in a local minimum.

The problem with this definition is that it still provides no guarantees about how greedy search will perform. For example, the clouds representing local minima in the graph shown in the top of Figure 5 are also local minima of the $h(n) - h^*(n)$ function, but as we previously showed, even if these local minima only contain one node, greedy search might still have arbitrarily poor performance if there are too many local minima, something independent of the size of a local minimum.

The graph in the bottom part of Figure 5 has a single large local minimum in the $h(n) - h^*(n)$ function. Despite this large local minimum, we have shown that greedy search exhibits nearly optimal run time, demonstrating that even in domains where the local minima in the $h(n) - h^*(n)$ function contains many nodes, greedy search may nonetheless be an effective algorithm.

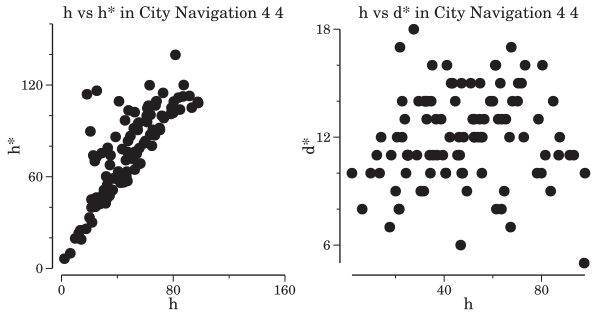

Figure 6: Plot of h vs h*, and h vs d* for City Navigation 4 4

We have shown that greedy search can perform arbitrarily poorly in domains with small local minima in the $h(n) - h^*(n)$ function, and also that greedy search can exhibit linear run time even if the local minima in the $h(n) - h^*(n)$ contains infinitely many nodes. This leads us to conclude that the size of the local minima in the $h(n) - h^*(n)$ function cannot be used to predict whether or not greedy search will fail.

Hypothesis 4 *Greedy search performs poorly when the correlation between $h(n)$ and $h^*(n)$ is weak.*

While considering Hypothesis 1, we noted that greedy search has run time linear in the solution length of the optimal solution if the nodes are in $h^*(n)$ order. Looking at the plot of $h(n)$ vs $h^*(n)$ in the left half of Figure 6, we can see that for City Navigation 4 4 there is a reasonable relationship between $h(n)$ and $h^*(n)$, in that the nodes with low

$h(n)$ tend to have small $h^*(n)$ values. One way to quantify this observation is to measure the correlation between the two values.

Pearson's correlation coefficient measures how well the relationship between $h(n)$ and $h^*(n)$ can be modeled using a linear function. Such a relationship would mean that weighting the heuristic appropriately can reduce the error in the heuristic, which could reasonably be expected to lead to a faster search. In addition, if the relationship between $h(n)$ and $h^*(n)$ is a linear function, then order will be preserved: putting nodes in order of $h(n)$ will also put the nodes in order of $h^*(n)$, which leads to an effective greedy search. For each domain, we calculate the Pearson's correlation coefficient between $h^*(n)$ and $h(n)$, and the results are in the second column of Table 2.

Another reasonable way to measure the heuristic correlation is to use Spearman's rank correlation coefficient. Spearman's rank correlation coefficient measures how well one variable can be modeled as a perfect monotone function of the other. In the context of greedy search, if Spearman's rank correlation coefficient is high, this means that the $h(n)$ and $h^*(n)$ put nodes in very close to the same order. Expanding nodes in $h^*(n)$ order leads to greedy search running in time linear in the solution length, so it is reasonable to conclude that a strong Spearman's rank correlation coefficient between $h^*(n)$ and $h(n)$ would lead to an effective greedy search. For each domain, we calculate the Spearman's rank correlation coefficient between $h^*(n)$ and $h(n)$, and the results are in the third column of Table 2.

Looking to the data, leads us to reject this hypothesis. In the TopSpin(3) domain, the Spearman's rank correlation coefficient is .52, so this hypothesis leads us to conclude that greedy search will work well in domains having a Spearman's rank correlation of .52 or higher. In the inverse tiles domain, City Navigation 3 3, and City Navigation 4 4 the Spearman's rank correlation coefficient is higher than .52, but in these domains, greedy search performs poorly. A similar argument holds for the Pearson's correlation coefficient.

Hypothesis 5 *Greedy search performs poorly when the correlation between $h(n)$ and $d^*(n)$ is weak.*

The objective of greedy search is to discover a goal quickly by expanding nodes with a small $h(n)$ value. If nodes with a small $h(n)$ are far away from a goal, and therefore have a high $d^*(n)$ value (high count of edges to goal), it is reasonable to believe greedy search would perform poorly. The right half of Figure 6 shows a plot of $h(n)$ vs $d^*(n)$. We can clearly see that in the City Navigation 4 4 domain, there is almost no relationship between $h(n)$ and $d^*(n)$, meaning that nodes that receive a small $h(n)$ value can be found any distance away from a goal, which could explain why greedy search works so poorly for this domain.

For each domain, we quantify this concept by calculating both Pearson's correlation coefficient and Spearman's rank correlation coefficient between $d^*(n)$ and $h(n)$, and the results are in the fourth and fifth column of Table 2. If we divide domains based upon either correlation metric considering the correlation between $h(n)$ and $d^*(n)$, we are finally able to differentiate between the domains where greedy search is effective and the domains where greedy search performs poorly.

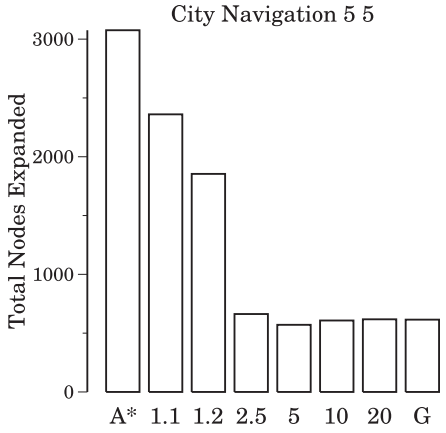

Figure 7: Expansions with differing number of neighbors for cities and places

Intuitively, this is reasonable: greedy search expands nodes with small $h(n)$ values. If nodes with small $h(n)$ value are also likely to have a small $d^*(n)$ value (and these nodes are therefore close to a goal, in terms of expansions away) expanding nodes with small $h(n)$ value will quickly lead to a goal. The converse is also reasonable. If the nodes with small $h(n)$ value have a uniform distribution of $d^*(n)$ values (and many of these nodes are far away from a goal in terms of expansions away), expanding these nodes will not quickly lead to a goal.

Hypothesis 5 neatly predicts when greedy search performs worse than Weighted A* (or A*). It is not, however, perfect. If we consider the heuristic $h(n) = h^*(n)$, any measure of the correlation between $h(n)$ and $h^*(n)$ will be perfect, but the relationship between $h(n)$ and $d^*(n)$ for this heuristic can be arbitrarily poor. As the heuristic approaches truth, the $h(n)$-$h^*(n)$ correlations will approach 1, which allows Weighted A* to scale gracefully, as greedy search will have linear run time, no matter what the correlation between $h(n)$ and $d^*(n)$ is. In this situation, looking solely to the correlation between $h(n)$ and $d^*(n)$ to determine whether or not greedy search will be faster than Weighted A* may produce an incorrect answer.

This can be seen in the City Navigation 5 5 domain. City Navigation 5 5 is exactly like the other City Navigation problems we consider, except that the cities and places are better connected, allowing more direct routes to be taken. Since the routes are more direct, and thus shorter, the heuristic is more accurate.

Table 3 shows the various correlations and percent error in $h(n)$ for City Navigation 5 5. Figure 7 shows that as we increase the weight, despite the very weak correlation between $h(n)$ and $d^*(n)$, there is no catastrophe: greedy search expands roughly the same number of nodes as Weighted A* with the best weight for speed. This occurs because of the extreme strength of the heuristic, which correlates to $h^*(n)$ at .95, an extremely strong correlation.

Discussion

The importance of the correlation between $h(n)$ and $d^*(n)$ shows the importance of node ordering for greedy search. In optimal search, the search cannot terminate when a solution is found, but rather when the solution is known the be optimal because all other paths have been pruned. The larger the heuristic values, the sooner nodes can be pruned. This means that in optimal search, heuristic size is of paramount importance: bigger is better. With greedy search, the heuristic is used to guide the search to a solution, so relative magnitude of the heuristic (or the error in the heuristic) has no bearing on the performance of the search, as we saw when we considered Hypothesis 1.

Effective node ordering, which can be measured directly by calculating Spearman's rank correlation coefficient, or indirectly by calculating Pearson's correlation coefficient, tells us how effectively the heuristic can be used to order nodes in a greedy search. The next question is which correlation matters more: $h^*(n)$ or $d^*(n)$. Clearly, a perfect correlation between $h^*(n)$ and $h(n)$ or $d^*(n)$ and $h(n)$ will lead to a fast greedy search, which leads us to the conclusion that in order for greedy search to be effective, nodes with small $h(n)$ that get expanded are required to have at least one virtue: they should either be close to the goal measured in terms of search distance (small $d^*(n)$) or close to the goal measured in terms of graph distance (small $h^*(n)$). We have seen empirically that as the two correlations break down, the $d^*(n)$ correlation allows greedy search to survive longer: in domains where the $d^*(n)$-$h(n)$ is above .58, greedy search does well. In domains where the $h^*(n)$-$h(n)$ correlation is as high as .70 (or .75, depending on which correlation metric is being used), we have domains where greedy search performs poorly.

Conclusion

We first showed that greedy search can sometimes perform worse than A*, and that although in many domains there is a general trend where a larger weight in Weighted A* leads to a faster search, there are also domains where a larger weight leads to a slower search. It has long been understood that greedy search has no bounds on performance, but our work shows that poor behavior can occur in practice.

We then showed that the domains where increasing the weight degrades performance share a common trait: the true distance from a node to a goal, defined as $d^*(n)$, correlates very poorly with $h(n)$. This information is important for anyone running suboptimal search in the interest of speed, because it allows them to identify whether or not the assumption that weighting speeds up search is true or not, critical knowledge for deciding what algorithm to use.

Acknowledgements

We acknowledge support from NSF (grant IIS-0812141) and the DARPA CSSG program (grant HR0011-09-1-0021).

References

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

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

Gaschnig, J. 1977. Exactly how good are heuristics?: Toward a realistic predictive theory of best-first search. In *Proceedings of the Fifth International Joint Conference on Articial Intelligence*, 434–441.

Helmert, M., and Röger, G. 2007. How good is almost perfect? In *Proceedings of the ICAPS-2007 Workshop on Heuristics for Domain-independent Planning: Progress, Ideas, Limitations, Challenges*.

Helmert, M. 2006. The fast downward planning system. *Journal of Artificial Intelligence Research* 26:191–246.

Helmert, M. 2010. Landmark heuristics for the pancake problem. In *Proceedings of the Third Symposium on Combinatorial Search*.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14:253–302.

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

Korf, R., and Felner, A. 2002. Disjoint pattern database heuristics. *Artificial Intelligence* 134:9–22.

Korf, R. E.; Reid, M.; and Edelkamp, S. 2001. Time complexity of iterative-deepening-A*. *Artificial Intelligence* 129:199–218.

Korf, R. E. 1993. Linear-space best-first search. *Artificial Intelligence* 62:41–78.

Lelis, L.; Zilles, S.; and Holte, R. C. 2011. Improved prediction of IDA*'s performance via epsilon-truncation. In *SOCS*.

Likhachev, M.; Gordon, G.; and Thrun, S. 2003. ARA*: Anytime A* with provable bounds on sub-optimality. In *Proceedings of the Seventeenth Annual Conference on Neural Information Processing Systems*.

Likhachev, M. 2005. *Search-based Planning for Large Dynamic Environments*. Ph.D. Dissertation, Carnegie Mellon University.

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

Pohl, I. 1970. Heuristic search viewed as path finding in a graph. *Artificial Intelligence* 1:193–204.

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

Richter, S.; Thayer, J. T.; and Ruml, W. 2009. The joy of forgetting: Faster anytime search via restarting. In *Proceedings of the Twentieth International Conference on Automated Planning and Scheduling*.

van den Berg, J.; Shah, R.; Huang, A.; and Goldberg, K. Y. 2011. Anytime nonparametric A*. In *Proceedings of the Twenty Fifth National Conference on Articial Intelligence*.