Journal of Artificial Intelligence Research 57 (2016) 273-306 Submitted 11/15; published 10/16

Effective Heuristics for Suboptimal Best-First Search

Christopher Wilt WILT AT CS.UNH.EDU
Wheeler Ruml RUML AT CS.UNH.EDU
Department of Computer Science
University of New Hampshire
Durham, NH 03824 USA

Abstract

Suboptimal heuristic search algorithms such as weighted A* and greedy best-first search are widely used to solve problems for which guaranteed optimal solutions are too expensive to obtain. These algorithms crucially rely on a heuristic function to guide their search. However, most research on building heuristics addresses optimal solving. In this paper, we illustrate how established wisdom for constructing heuristics for optimal search can fail when considering suboptimal search. We consider the behavior of greedy best-first search in detail and we test several hypotheses for predicting when a heuristic will be effective for it. Our results suggest that a predictive characteristic is a heuristic's goal distance rank correlation (GDRC), a robust measure of whether it orders nodes according to distance to a goal. We demonstrate that GDRC can be used to automatically construct abstraction-based heuristics for greedy best-first search that are more effective than those built by methods oriented toward optimal search. These results reinforce the point that suboptimal search deserves sustained attention and specialized methods of its own.

1. Introduction

A* is a best-first search that expands nodes in order of $f(n)$ where $f(n) = g(n) + h(n)$. While the optimal solutions provided by A* (Hart, Nilsson, & Raphael, 1968) are the most desirable, time and memory often prevent the application of this algorithm. When A* fails because of either insufficient time or memory, practitioners sometimes turn to bounded suboptimal algorithms that may not return the optimal solution, but that return a solution that is guaranteed to be no more than a certain factor more expensive than the optimal solution.

The most well-known of these is likely Weighted A* (Pohl, 1970), which is a best-first search that expands nodes in f' order, where $f'(n) = g(n) + w \cdot h(n) : w \in (1, \infty)$. Variants of Weighted A* are used in a wide variety of applications, including domain-independent planning (Helmert, 2006; Richter & Westphal, 2010) and robotics (Likhachev, Gordon, & Thrun, 2003; Likhachev & Ferguson, 2009). Weighted A* is also a component of a number of anytime algorithms. For example, Anytime Restarting Weighted A* (Richter, Thayer, & Ruml, 2009) and Anytime Repairing A* (Likhachev et al., 2003) both use Weighted A*. Anytime Nonparametric A* (van den Berg, Shah, Huang, & Goldberg, 2011) doesn't use Weighted A* per se, but rather its limiting case, greedy best-first search (Doran & Michie, 1966), best-first search on $h(n)$. All of these anytime algorithms have, built in, the implicit assumption that Weighted A* with a high weight or greedy best-first search will find a solution faster than A* or Weighted A* with a small weight.

In many popular heuristic search benchmark domains (e.g., sliding tile puzzles, grid path planning, Towers of Hanoi, TopSpin, robot motion planning and the traveling salesman problem) 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 best-first search, which results in the fastest search. The first contribution of this paper is to provide illustrations of how, in some domains, greedy best-first search performs worse than Weighted A*, and is sometimes even worse than A*.

We show that the failure of greedy best-first search is not merely a mathematical curiosity, only occurring in hand crafted counterexamples, but rather a phenomenon that can occur in real domains, including variants of popular single-agent heuristic benchmarks. Our second contribution is to empirically characterize conditions when this occurs, knowledge that 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.

The root cause of the failure of greedy best-first search can be ultimately traced back to the heuristic, which is used to guide a greedy best-first search to a goal. For A*, there are a number of well-documented techniques for constructing an effective heuristic. We revisit these guidelines in the context of greedy best-first search. Our third contribution is to show that, if one follows the well-established guidelines for creating a quality heuristic for A*, the results can be poor. We present several examples where following the A* wisdom for constructing a heuristic leads to slower results for greedy best-first search. We use these examples to understand the requirements that greedy best-first search places on its heuristic.

Our fourth contribution is a quantitative metric for assessing a greedy heuristic, goal distance rank correlation (GDRC). GDRC can be used to predict whether or not greedy best-first search is likely to perform well. GDRC can also be used to compare different heuristics for the same domain, allowing us to make more informed decisions about which heuristic to select if there are a variety of choices, as is the case for abstraction-based heuristics like pattern databases. This quantitative metric can be used to automatically construct a heuristic for greedy best-first search by iteratively refining an abstraction and measuring how good each candidate heuristic is. We show that iteratively refining an abstraction using a simple hill-climbing search guided by GDRC can yield heuristics that are more powerful than those built by traditional methods oriented toward optimal search.

This work increases our understanding of greedy best-first search, one of the most popular and scaleable heuristic search techniques. More generally, it suggests that techniques developed for optimal search are not necessarily appropriate for suboptimal search. Suboptimal search is markedly different from optimal search, and deserves its own theory and methods.

2. A Conundrum: Ineffective Weighted A*

The starting point for our investigation of heuristics for suboptional search begins with a curious empirical observation: although weighted A* is one of the most popular way of speeding up heuristic search, increasing the weight of Weighted A* does not always work. In order to get a better grasp on the question of when increasing the weight is ineffective, we first need some empirical data.

Domain	Average Solution Length	Total States	Branching Factor	Unit-cost
Dynamic Robot	187.45	20,480,000	0-240	No
Hanoi (14)	86.92	268,435,456	6	Yes
Pancake (40)	38.56	8×10^{47}	40	Yes
11 Tiles (unit)	36.03	239,500,800	1-3	Yes
Grid	2927.40	1,560,000	0-3	Yes
TopSpin (3)	8.52	479,001,600	12	Yes
TopSpin (4)	10.04	479,001,600	12	Yes
11 Tiles (inverse)	37.95	239,500,800	1-3	No
City Navigation 3 3	15.62	22,500	3-8	No
City Navigation 4 4	14.38	22,500	3-10	No
City Navigation 5 5	13.99	22,500	3-12	No

Table 1: Domain Attributes for benchmark domains considered

2.1 Benchmark Domains

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. 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. Since we would like to compare against A*, 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 best-first search in main memory (eight gigabytes). Basic statistics about each of these domain variants are summarized in Table 1.

For the sliding tile 11 puzzle (3×4), we used random instances and the Manhattan distance heuristic. We used the 11 puzzle, rather than the 15 puzzle for two reasons. First, optimally solving 15 puzzles using A* without running out of memory requires significant resources (At least 27 gigabytes, significantly more than our eight gigabyte limit, according to Burns et al., 2012). In addition to that, we consider the sliding tile puzzle with non-unit cost functions. These non-unit problems are significantly more difficult to solve than the unit-cost variants. The non-unit version of sliding tile puzzle we consider uses the inverse cost function, where the cost of moving a tile n is $1/n$. The Manhattan distance heuristic, when weighted appropriately, is both admissible and consistent for this cost function. 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 & 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. In the TopSpin puzzle, the objective is to sort a circular permutation by iteratively reversing a continuous subsequence of fixed size. An example of a TopSpin puzzle is in Figure 1. We considered a problem with 12 disks with a turnstile that would turn either three or four disks, denoted by TopSpin(3) and TopSpin(4). For a heuristic, we used a pattern database with 6 contiguous disks present,

Figure 1: A 20 disk TopSpin puzzle.

and the remaining 6 disks abstracted. For the dynamic robot navigation problem, we used a 200x200 world, with 32 headings and 16 speeds. In dynamic robot navigation, the objective is to navigate a robot from one location and heading to another location and heading, while respecting the dynamics of the robot. The robot is not able to change direction and speed instantaneously, so not all combinations of heading/speed can be reached from a given state. In addition to that, some states in this domain represent dead ends. For example, a state where the robot is moving at full speed directly towards an obstacle will produce no children, because the robot will crash no matter what control action is applied. The objective is to minimize the total travel time; the actions do not all have the same cost.

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 which guarantees it is possible to get from any city to any other city. 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 connected in a random tour, with each place also 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 there 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 to reach the goal in the example problem we must 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. An example instance of this type can be seen in Figure 2. The circles in the left part of the figure are locations, connected to other locations. The nexus node, Location 0, is also connected to the nexus nodes of neighboring

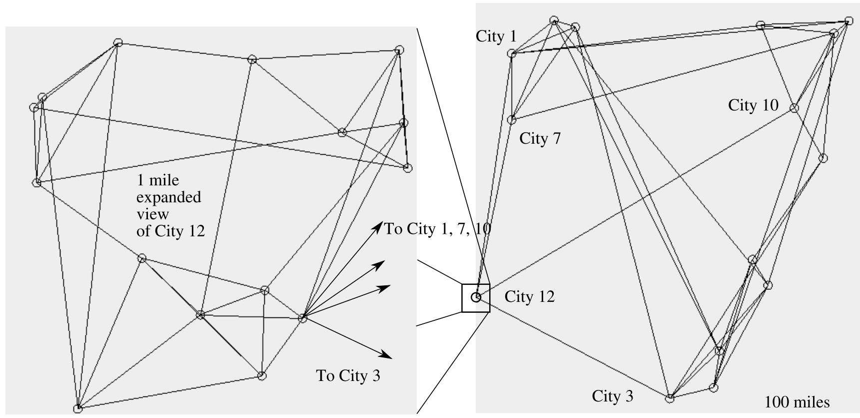

Figure 2: A city navigation problem with $n_p = n_c = 3$, with 15 cities and 15 locations in each city.

cities. The right part of the figure shows the entire world, with cities shrunk down to a circle.

City Navigation instances are classified by n_p and 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. In this domain, solutions vary in length, and it is straightforward to manipulate the accuracy of the heuristic. This domain bears some similarity to the IPC Logistics domain in which locations within cities are connected by roads, but special airport locations are used to travel between cities.

2.2 Results

Figures 3 and 4 show the number of expansions required by A*, greedy best-first search, and Weighted A* with weights of 1.1, 1.2, 2.5, 5, 10, and 20. These plots allow us to compare greedy best-first 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 3, it is easy to see that as we increase the weight the number of expansions goes down, but in Figure 4, 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 best-first 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 best-first search is higher than the number of nodes expanded by A*. Explaining this phenomenon is a central goal of this paper.

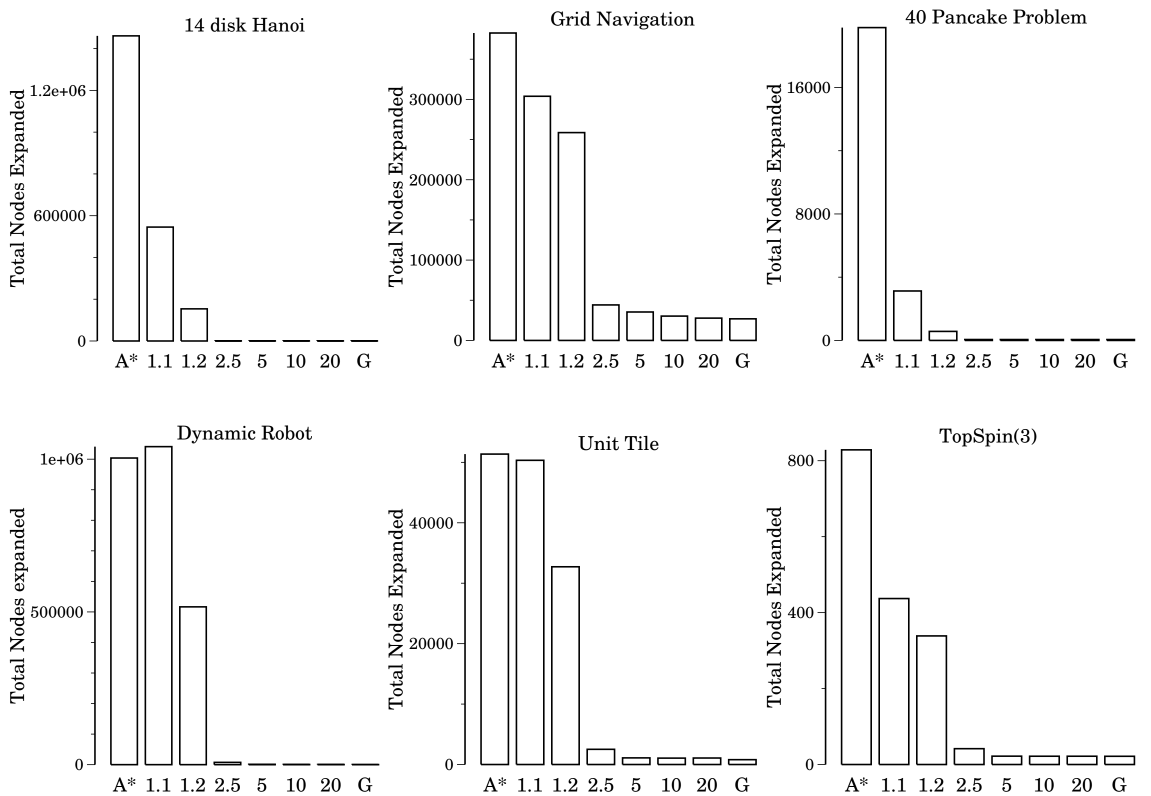

Figure 3: Domains where increasing the weight speeds up search. Numbers denote Weighted A* run with a specific weight, and G denotes greedy best-first search.

3. Characteristics of Effective Heuristics

We have established that increasing the weight in Weighted A* does not always speed up the search, and in some situations can actually slow down search. The fact that A* is sometimes faster than greedy best-first search and sometimes slower than greedy best-first search suggests that some heuristics work well for A* and poorly for greedy best-first search, and that some heuristics work well for greedy best-first search but not for A*. Thus, the question is precisely what is driving this difference, and what each algorithm, A* and greedy best-first search, needs out of the heuristic.

We first review the literature for suggestions about how to make a good heuristic for A*. With this in mind, we then apply the A* rules for constructing an effective heuristic to greedy best-first search. This leads us to observations on effective heuristics for greedy best-first search that are distinct from the common recommendations for building a good heuristic for A*.

3.1 Effective Heuristics for A*

Much of the literature about what constitutes a good heuristic centers on how well the heuristic works for A*. For finding optimal solutions using A*, the first and most important

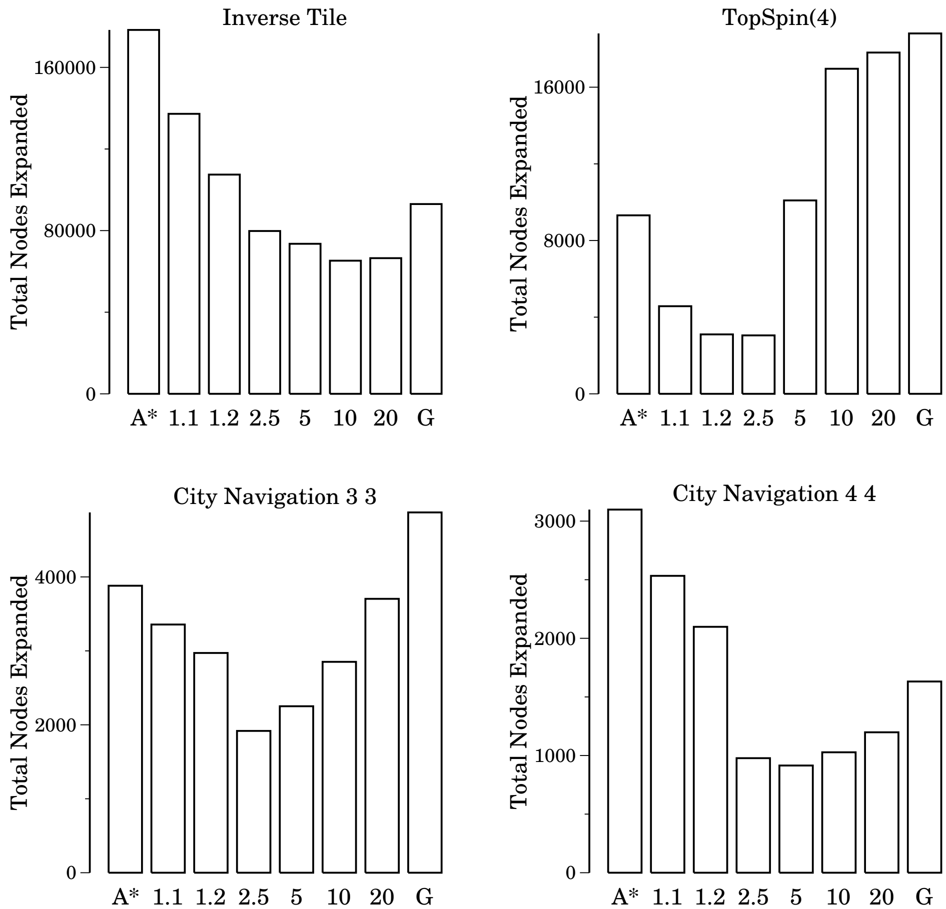

Figure 4: Domains where increasing the weight slows down search. Numbers denote Weighted A* with the specified weight, and G denotes greedy best-first search.

requirement is that the heuristic be admissible, meaning for all nodes n, $h^*(n)$ – the true cheapest path from n to a goal – is greater than or equal to $h(n)$. If the heuristic is not admissible, A* degenerates into A (no star) which is not guaranteed to find the shortest path.

It is generally believed that consistency is also important, due to the fact that inadmissible heuristics can lead to an exponential number of re-expansions (Martelli, 1977). This situation, however, rarely arises in practice and Felner et. al. (2011) argue that inconsistency is generally not as much of a problem as is generally believed.

The most widespread rule for making a good heuristic for A* is: dominance is good (Nilsson, 1980; Pearl, 1984). A heuristic h_1 is said to dominate h_2 if $\forall n \in G : h_1(n) \geq h_2(n)$. This makes sense, because due to admissibility, larger values are closer to h^*. Furthermore A* must expand every node n it encounters where $f(n)$ is less than the cost of an optimal solution, so large h often reduces expansions. Dominance represents the current gold standard for comparing two heuristics. In practice, heuristics are often informally evaluated by their average value or by their value at the initial state over a benchmark set. In either case, the general idea remains the same: bigger heuristics are better.

If we ignore the effects of tie breaking as well as the effects of duplicate states, A* and the last iteration of IDA* expand the same number of nodes. This allows us to apply the formula from Korf, Reid, and Edelkamp (2001). They predict that the number of nodes IDA* will expand at cost bound c is:

$$E(N, c, P) = \sum_{i=0}^{c} N_i P(c - i)$$

The function $P(h)$ in the KRE equation represents the equilibrium heuristic distribution, which is "the probability that a node chosen randomly and uniformly among all nodes at a given depth of the brute-force search tree has heuristic value less than or equal to h" (Korf et al., 2001). This quantity tends to decrease as h gets larger, depending on how the nodes in the space are distributed. The dominance relation also transfers to the KRE equation, meaning that if a heuristic h_1 dominates a different heuristic h_2, the KRE equations predicts that the expected expansions using h_1 will be less than or equal to the expected expansions using h_2.

When considering pattern database (PDB) heuristics, Korf's conjecture (1997) can lend insight into the performance of IDA*, which tells us that we can expect $\frac{m}{1+log(m)} \times t = n$ with m being the amount of memory the PDB in question takes up, t is the amount of time we expect an IDA* search to consume, and n is a constant (Korf, 2007). If we are willing to apply results regarding IDA* to A* this equation tells us that we should expect larger pattern databases to provide faster search for A*. To summarize, the prevailing wisdom regarding heuristics is that bigger is better, both in terms of average heuristic value and pattern database size.

3.2 The Behavior of Greedy Best-First Search

As we shall see, this advice regarding heuristics is all very helpful when considering only A*. What happens if we apply this same wisdom to greedy best-first search? We answer this question by taking a detailed look at the behavior of greedy best-first search on three of our benchmark problems: the Towers of Hanoi, the TopSpin puzzle, and the sliding tile puzzle.

3.2.1 Towers of Hanoi

The first domain we consider is the Towers of Hanoi. The most successful heuristic for optimally solving 4 peg Towers of Hanoi problems is disjoint pattern databases (Korf & Felner, 2002). Disjoint pattern databases boost the heuristic value by providing information about the disks on the top of the puzzle. For example, consider a 12-disk puzzle, split into two disjoint pattern databases: eight disks in the bottom pattern database, and four disks in the top pattern database. With A*, the best results are achieved when using the full disjoint pattern database. With greedy best-first search, however, faster search results when we do not use a disjoint pattern database, and instead only use the 8 disk pattern database. The exact numbers are presented in the Unit rows of Table 2. All problems are randomly generated Towers of Hanoi states, with the goal being to get all disks onto the first peg.

Cost	Heuristic	A* Exp	Greedy Exp
Unit	8/4 PDB	2,153,558	36,023
	8/0 PDB	4,618,913	771
Square	8/4 PDB	239,653	4,663
	8/0 PDB	329,761	892
Rev Square	8/4 PDB	3,412,080	559,250
	8/0 PDB	9,896,145	730

Table 2: Average number of nodes expanded to solve 51 12-disk Towers of Hanoi problems.

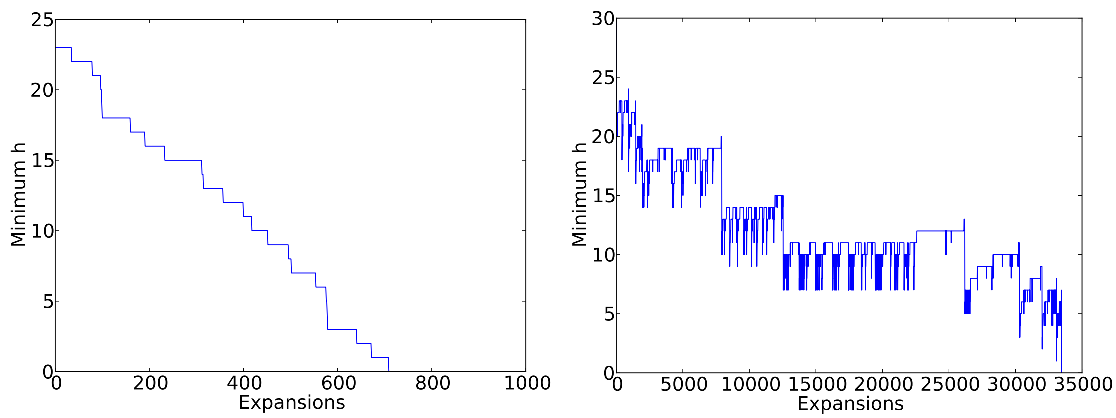

Figure 5: The minimum h value on open as the search progresses, using different pattern databases (single on left, two disjoint additive ones on the right).

The theory for A* corroborates the empirical evidence observed here: the disjoint pattern database dominates the single pattern database, so absent unusual effects from tie-breaking, it is no surprise that the disjoint pattern database results in faster A* search.

The reason for the different behaviour of A* and greedy best-first search is simple. With greedy best-first search using a single pattern database, it is possible to follow the heuristic directly to a goal, having the h value of the head of the open list monotonically decrease. To see this, note that every combination of the bottom disks has an h value, and all possible arrangements of the disks on top will also share that same h value. The disks on top can always be moved around independently of where the bottom disks are. Consequently, it is always possible to arrange the top disks such that the next move of the bottom disks can be done, while not disturbing any of the bottom disks, thus leaving h constant. Eventually, h decreases because more progress has been made putting the bottom disks of the problem in order. This process repeats until $h = 0$, at which point greedy best-first search simply considers possible configurations of the top disks until the goal has been found.

This phenomenon can be seen in the left pane of Figure 5, where the minimum h value of the open list monotonically decreases as the number of expansions the search has done

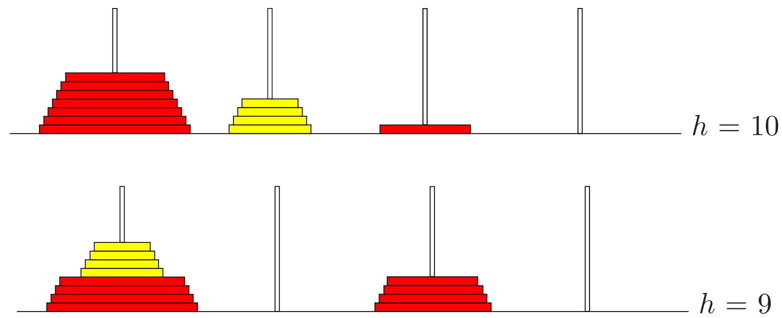

Figure 6: Two Towers of Hanoi states, one near a goal (top) and one far from a goal (bottom).

increases. The heuristic created by the single pattern database creates an extremely effective gradient for the greedy best-first search algorithm to follow for two reasons. First, there are no local minima at all, only the global minimum where the goal is. In this context, we define a minimum as a region of the space M where $\forall n \in M$, every path from n to a goal node has at least one node n' with $h(n') > h(n)$. Second, there are exactly 256 states associated with each configuration of the bottom 8 disks. This means that every 256 expansions, h is guaranteed to decrease. In practice, a state with a lower h tends to be found much faster.

In the right pane of Figure 5, the heuristic is a disjoint pattern database. We can see that the h value of the head of the open list fluctuates substantially when using a disjoint pattern database, indicating that greedy best-first search's policy of "follow small h" is much less successful. This is because those states with the bottom disks very near their goal that are paired with a very poor arrangement of the disks on top are assigned large heuristic values, which delays the expansion of these nodes. This is illustrated in Figure 6. The top state is significantly closer to a goal, despite having a higher h value than the bottom state. If we ignore the top disks completely, the top state has $h = 1$ compared to the bottom state's $h = 9$, which correctly conveys the fact that the top state is significantly closer to a goal. The disjoint PDB causes substantial confusion for greedy best-first search, because prior to making any progress with any of the 8 bottom disks, the greedy best-first search considers states where the top 4 disks are closer to their destination. If the bottom state is expanded, it will produce children with lower heuristic values which will be explored before ever considering the top state, which is the state that should be explored first. Eventually, all descendants of the bottom state with $h \leq 9$ are explored, at which point the top state is expanded, but this causes the h value of the head of the open list to go up and down.

To summarize, the disjoint pattern database makes a gradient that is more difficult for greedy best-first search to follow because nodes can have a small h for more than one reason: being near the goal because the bottom pattern database is returning a small value, or being not particularly near the goal, but having the top disks arranged on the target peg. This suggests the following observation regarding heuristics for greedy best-first search:

Observation 1. *All else being equal, greedy best-first search tends to work well when it is possible to reach the goal from every node via a path where h monotonically decreases along the path.*

While this may seem self-evident, our example has illustrated how it conflicts with the common wisdom in heuristic construction. It is also important to note that this observation makes no comment about the relative magnitude of the heuristic, which for greedy best-first is completely irrelevant; all that matters is the relative ordering of the nodes when ordered using the heuristic.

Another way to view this phenomenon is in analogy to the Sussman Anomaly (Sussman, 1975). The Sussman anomaly occurs when one must undo a subgoal prior to being able to reach the global goal. In the context of Towers of Hanoi problems, the goal is to get all of the disks on the target peg, but solving the problem may involve doing and then undoing some subgoals of putting the top disks on the target peg. The presence of the top pattern database encourages greedy best-first searches to privilege states where subgoals which eventually have to be undone have been accomplished.

Korf (1987) discusses different kinds of subgoals, and how different kinds of heuristic searches are able to leverage subgoals. Greedy best-first search uses the heuristic to create subgoals, attempting to follow the h to a goal. For example, in a unit-cost domain, the first subgoal is to find a node with $h = h(root) - 1$. If the heuristic follows Observation 1, these subgoals form a perfect serialization, and the subgoals can be achieved one after another. As the heuristic deviates from Observation 1, the subgoals induced by the heuristic cannot be serialized.

Another important factor is, of course, the number of distinct nodes at each heuristic level one encounters prior to finding a better node. Consider, for example, one of the worst heuristics, $h = 0$. Technically, this heuristic follows Observation 1 because all paths only contain nodes with $h = 0$, but the one plateau contains all nodes in the entire space, which is obviously undesirable. Hoffmann (2005) discusses this general idea using the term "maximal bench exit distance", and once again, the idea is that in domains in which this quantity is small, both greedy best-first search and his Enforced Hill Climbing method perform well, because finding nodes with lower h is straightforward.

These effects can be exacerbated if the cost of the disks on the top is increased relative to the cost of the disks on the bottom. If we define the cost of moving a disk as being proportional to the disk's size, we get the Square cost metric, where the cost of moving disk n is n^2. We could also imagine the tower being stacked in reverse, requiring that the larger disks always be on top of the smaller disks, in which case we get the Reverse Square cost function. In either case, we expect that the number of expansions that greedy best-first search will require will be lower when using only the bottom pattern database, and this is indeed the effect we observe in Table 2. However, if the top disks are heavier than the disks on the bottom, greedy best-first search suffers even more than when we considered the unit cost problem, expanding an order of magnitude more nodes. This is because the pattern database with information about the top disks is returning values that are substantially larger than the bottom pattern database, due to the fact that the top pattern database considers the most expensive operators. If the situation is reversed, however, and the top pattern database uses only the lowest cost operators, the top pattern database's contribution to h is a much smaller proportion of the total expansions. Since greedy best-first search

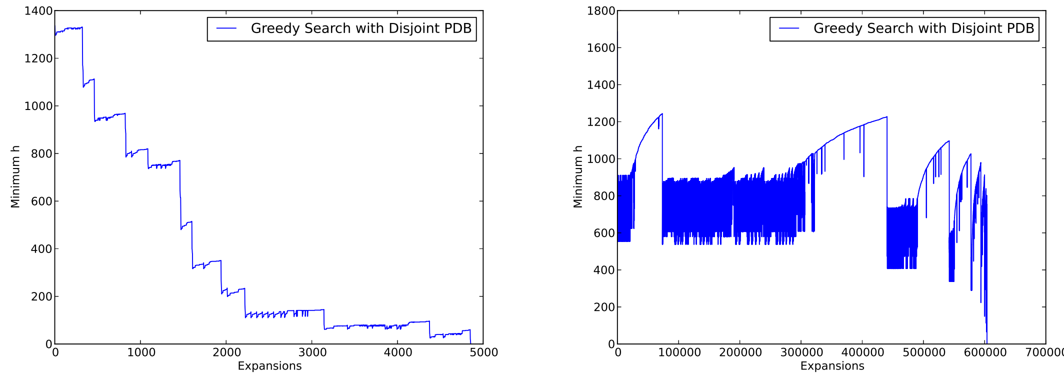

Figure 7: The minimum h value on open during searches using disjoint pattern databases with different cost functions (square on left, reverse square on right).

performs best when the top pattern database isn't even present, it naturally performs better when the contribution of the top pattern database is smaller.

This phenomenon is vividly illustrated in the execution times in Figure 7. In the left of the figure, the disks in the top pattern database are much cheaper to move than the disks in the bottom pattern database, and are therefore contributing a much smaller proportion of the total value of h. In the right part of the figure, the disks in the top pattern database are much more expensive to move than the disks in the bottom pattern database, so the top pattern database makes a much larger contribution to h, causing substantially more confusion.

Hoffmann (2005) notes that the success of the FF heuristic in many domains is attributable to the fact that the h^+ heuristic produces a heuristic with no local minima. A heuristic with no local minima precisely matches our Observation 1, because it will always be possible to reach the goal via a path where h monotonically decreases.

3.2.2 TopSpin

We considered TopSpin with 12 disks and a turnstile that flipped 4 disks using pattern databases that contained 5, 6, 7, and 8 of the 12 total disks.

Korf's conjecture predicts that the larger pattern databases will be more useful for A*, and should therefore be considered to be stronger heuristics, and indeed, as the PDB becomes larger, the number of expansions done by A* dramatically decreases. This can be seen in Figure 8. Each box plot (Tukey, 1977) is labeled with either A* or G (for greedy best-first search), and a number, denoting the number of disks that the PDB tracks. Each box denotes the middle 50% of the data, so the top of the box is the upper quartile, the bottom of the box is the bottom quartile, and the height of the box is the interquartile range. The horizontal line in the middle of the box represents the median. The grey stripe indicates the 95% confidence interval about the mean. The circles denote points that are more than 1.5 times the interquartile range away from either the first quartile or the third

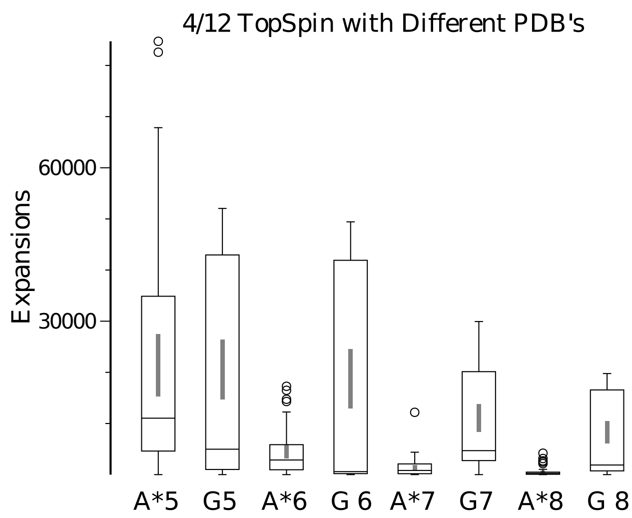

Figure 8: TopSpin puzzle with different heuristics. A* followed by a number denotes A* with that number of disks in the PDB heuristic. G followed by a number denotes greedy best-first search with that number of disks in the PDB heuristic.

quartile, and the whiskers represent the range of the non-outlier data. As we move from left to right, as the PDB heuristic tracks more disks, it gets substantially better for A*. While there are also reductions for greedy best-first search in terms of expansions, the gains are nowhere near as impressive as compared to A*.

The reason that greedy best-first search does not perform better when given a larger heuristic is that, with the larger heuristic, states with $h = 0$ may still be quite far from a goal. For example, consider the TopSpin state represented as follows, where A denotes an abstracted disk:

State 1: 0 1 2 3 4 5 A A A A A A

The turnstile swaps the orientation of 4 disks, but there are configurations such that putting the abstracted disks in order requires moving a disk that is not abstracted, such as:

State 2: 0 1 2 3 4 5 6 7 8 9 11 10

For a TopSpin state, the abstraction process takes the largest N disks and converts them to abstracted disks, and abstracted disks are all treated the same, so State 2 would be abstracted into State 1, which means that it abstracts to the same state as the goal, making its heuristic 0. If we wanted to expand State 2, we could do so and one of the children is State 3, whose heuristic is still 0:

State 3: 0 1 2 3 4 5 6 7 10 11 9 8

Consider a different child, for example the child obtained by rotating the middle 4 disks:

State 4: 0 1 2 3 7 6 5 4 8 9 10 11

which abstracts into:

State 5: 0 1 2 3 A A 5 4 A A A A

The heuristic for State 4 is not 0, because State 4 abstracts into State 5. State 5 is an abstract state that is different from State 1 (the abstracted goal) so the heuristic of State 4 is not 0.

If we abstract disks 6-11, we still have the same abstract state as before, so the heuristic is still 0. Moving a disk that is not abstracted will increase the heuristic, but moving only abstracted disks will leave the heuristic at 0. Unfortunately, transforming State 2 into a goal cannot be done without moving at least one of the disks whose index is between 0 and 5, because the turnstile is of size 4.

This means that the subgraph consisting of only nodes with $h = 0$ in the TopSpin problem is disconnected. Thus, when greedy best-first search encounters a state with $h = 0$, the state could be a $h = 0$ state that is connected to the goal via only $h = 0$ states, which would be desirable, or the state could be a $h = 0$ state that is connected to the goal via only paths that contain at least one $h \neq 0$ nodes, which would be undesirable. If this is the case, greedy best-first search will first expand all the $h = 0$ nodes connected to the first $h = 0$ node (which by hypothesis is not connected to a goal node via paths only containing $h = 0$ nodes), and will then return to expanding nodes with $h = 1$, looking to find a different $h = 0$ node.

The abstraction controls the number and size of $h = 0$ regions. For example, if we abstract 6 disks, there are two strongly connected regions of $h = 0$ nodes, each containing 360 nodes. If we instead abstract 5 disks, there are 12 strongly connected $h = 0$ regions, each with 10 nodes. For the heuristic that abstracts 6 disks, there is a 50% chance that any given $h = 0$ node is connected to the goal via only $h = 0$ nodes, but once greedy best-first search has entered the correct $h = 0$ region, finding the goal node is largely up to chance. For the heuristic that abstracts 5 disks, the probability that any given $h = 0$ node is connected to the goal via only $h = 0$ nodes is lower. Once the correct $h = 0$ region is found, however, it is much easier to find the goal, because the region contains only 10 nodes, as compared to 360 nodes. Empirically, we can see that these two effects roughly cancel one another out, because the total number of expansions done by greedy best-first search remains roughly constant no matter which heuristic is used. This brings us to our next observation.

Observation 2. *All else being equal, nodes with $h = 0$ should be connected to goal nodes via paths that only contain $h = 0$ nodes.*

One can view this as an important specific case of Observation 1. Interestingly, some types of heuristics, such as the delete-relaxation heuristics used in domain-independent planning, obey this observation implicitly by never allowing non-goal states to have h values of 0.

One obvious way to make a heuristic satisfy this recommendation is to change the heuristic for all non-goal states to be the same as the minimum cost operator from the domain with cost of ϵ. If we do this, we can simply restate the recommendation substituting ϵ for 0, and we arrive at a similar result.

	A	A	3
A	A	A	7
8	9	10	11

	1	A	3
4	A	6	A
A	9	A	11

Figure 9: Different tile abstractions. "A" denotes a tile that is abstracted..

Abstraction	Greedy Exp	A* Exp
Outer L (Figure 9 left)	258	1,251,260
Checker (Figure 9 right)	11,583	1,423,378
Outer L Missing 3	3,006	DNF
Outer L Missing 3 and 7	20,267	DNF
Instance Specific	8,530	480,250
GDRC Generated	427	1,197,789
Average 6-tile PDB	17,641	1,609,995
Worst 6-tile PDB	193,849	2,168,785

Table 3: Average number of expansions required by Greedy best-first search and A* to solve 3×4 tile instances with different pattern databases. DNF denotes at least one instance would require more than 8GB to solve.

3.2.3 Sliding Tiles

The sliding tile puzzle is one of the most commonly used benchmark domains in heuristic search. As such, this domain is one of the best understood. Pattern database heuristics have been shown to be the strongest heuristics for this domain, and have been the strongest heuristics for quite some time (Korf & Taylor, 1996; Felner, Korf, Meshulam, & Holte, 2007). We use the 11 puzzle (4×3) as a case study because the smaller size of this puzzle allows creating and testing hundreds of different pattern databases. The central problem when constructing a pattern database for a sliding tile puzzle is selecting a good abstraction.

The abstraction that keeps only the outer L, shown in the left part of Figure 9, is extremely effective for greedy best-first search, because once greedy best-first search has put all abstracted tiles in their proper places, all that remains is to find the goal, which is easy to do using even a completely uninformed search on the remaining puzzle, as there are only $\frac{6!}{2} = 360$ states with $h = 0$ and the $h = 0$ states form a connected subgraph. This is analogous to the heuristic directing the search algorithm to follow the process outlined by Parberry (1995), in which large sliding tile puzzles are solved by first solving the outer L, and then treating the remaining problem as a smaller sliding tile puzzle.

Compare this to what happens when greedy best-first search is run on a checkerboard abstraction, as shown in the right part of Figure 9. Once greedy best-first search has identified a node with $h = 0$, there is a very high chance that the remaining abstracted tiles are not configured properly, and that at least one of the non-abstracted tiles will have to be moved. This effect can be seen in Table 3, where the average number of expansions required by A* is comparable with either abstraction, while the average number of expansions required by greedy best-first search is larger by two orders of magnitude.

The sheer size of the PDB is not as important for greedy best-first search as it is for A*. In Table 3, we can see that as we weaken the pattern database by removing the 3 and 7 tiles, the number of expansions required increases by a factor of 10 for greedy best-first search. For A* using the PDB with the 3 tile missing, 3 instances are unsolvable within 8 GB of memory (approximately 25 million nodes in our Java implementation). With both the 3 and the 7 tile missing, A* is unable to solve 16 instances within the same limit. It is worth noting that even without the 3 tile, the outer L abstraction is still more effective for greedy best-first search as compared to the checkerboard abstraction.

The underlying reason behind the inefficiency of greedy best-first search using certain pattern databases is the fact that the less useful pattern databases have nodes with $h = 0$ that are nowhere near the goal. This provides further evidence in favor of Observation 2; greedy best-first search concentrates its efforts on finding and expanding nodes with a low h value, and if some of those nodes are, in reality, not near a goal, this clearly causes problems for the algorithm. Because A* uses f, and g contributes to f, A* is able to eliminate some of these states from consideration (not expand them) because the high g value helps to give the node a high f value, which causes A* to relegate the node to the back of the expansion queue.

The checkerboard pattern database also helps to make clear another problem facing greedy best-first search heuristics. Once the algorithm discovers a node with $h = 0$, if that node is not connected to any goal via only $h = 0$ nodes, the algorithm will eventually run out of $h = 0$ nodes to expand, and will begin expanding nodes with $h = 1$. When expanding $h = 1$ nodes, greedy best-first search will either find more $h = 0$ nodes to examine for goals, or it will eventually exhaust all of the $h = 1$ nodes as well, and be forced to consider $h = 2$ nodes. A natural question to ask is how far the algorithm has to back off before it will be able to find a goal. This leads us to our next observation.

Observation 3. *All else being equal, greedy best-first search tends to work well when the difference between the minimum h value of the nodes in a local minimum and the minimum h that will allow the search to escape from the local minimum and reach a goal is low.*

This phenomenon is clearly illustrated when considering instance-specific pattern databases (Holte, Grajkowskic, & Tanner, 2005). In an instance-specific pattern database, the tiles that start out closest to their goals are abstracted first, leaving the tiles that are furthest away from their goals to be represented in the pattern database. This helps to maximize the heuristic values of the states near the root, but due to consistency this can also have the undesirable side effect of making states that are required to be included in a path to the goal have high heuristic values as well. Raising the heuristic value of the initial state is helpful for A* search, as evidenced by the reduction in the number of expansions for A* using instance-specific abstractions of the same size, shown in Table 3. Unfortunately, this approach is still not as powerful for greedy best-first search as the simpler outer L abstraction, or even the smaller variant missing the 3. This is because some instance-specific pattern databases use patterns that are difficult for greedy best-first search to use effectively, similar to the problems encountered when using the checkerboard abstraction.

	Domain	Heuristic % Error	$h(n)$-$h^*(n)$ Correlation (Pearson)	$h(n)$-$h^*(n)$ Correlation (Spearman)	$h(n)$-$h^*(n)$ Correlation (Kendall)
Greedy Works	Towers of Hanoi	29.47	0.9652	0.9433	0.8306
	Grid	25.11	0.9967	0.9958	0.9527
	Pancake	2.41	0.9621	0.9593	0.9198
	Dynamic Robot	15.66	0.9998	0.9983	0.9869
	Unit Tiles	33.37	0.7064	0.7065	0.5505
	TopSpin(3)	25.95	0.5855	0.4598	0.4158
Greedy Fails	TopSpin(4)	32.86	0.2827	0.3196	0.2736
	Inverse Tiles	29.49	0.6722	0.6584	0.4877
	City Nav 3 3	44.51	0.5688	0.6132	0.4675
	City Nav 4 4	37.41	0.7077	0.7518	0.6238

Table 4: Average % error and correlation between $h(n)$ and $h^*(n)$

4. Predicting Effectiveness of Greedy Heuristics

In the previous section, we saw that common wisdom regarding effective heuristics for optimal search did not carry over to suboptimal search. Instead, our examples motivated three general observations regarding what greedy best-first search looks for in a heuristic. While these qualitative observations are perhaps helpful heuristics for heuristic design, it is also useful to have a simple, quantitative metric for evaluating and comparing heuristics.

We begin by considering two intuitively reasonable quantitative metrics, the percent error in h, and the correlation between h and h^*. For each of these metrics, we show that the metric cannot be used to predict whether or not greedy best-first search will perform worse than Weighted A*. Then we consider a measure of "search distance to go" called d^*. $d^*(n)$ is the same as h^* if we change the graph by making all edges cost 1. We find that the correlation between h and d^* can be used to predict when greedy best-first search will perform poorly.

4.1 Percent Error in $h(n)$

The first metric we consider is perhaps the most intuitive measure of heuristic performance: the percent error in h. We define the percent error in the heuristic as $\frac{h^*(n)-h(n)}{h^*(n)}$. Since greedy best-first 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 best-first search does, is not going to lead to a fast search.

In Table 4, 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 best-search will be a poor choice. Towers of Hanoi, unit tiles, and TopSpin(3), three domains where greedy best-first search is effective, have as much or more heuristic error than domains where greedy best-first search works poorly. This leads us to conclude that you cannot measure the average heuristic percent error and use this to predict whether or not increasing the weight will speed up or slow down search.

To see intuitively why this makes sense, note that greedy best-first search only really requires that the nodes get put in $h^*(n)$ order by the heuristic. The exact magnitude, and therefore error, of the heuristic is unimportant, but magnitude has a huge effect on the average percent error. This can be seen if we consider the heuristic $h(n) = \frac{h^*(n)}{R} : R \in \mathbb{R}^+$ for a very large or very tiny R, which will always guide greedy best-first search directly to an optimal goal, while exhibiting arbitrarily high average percent error in the heuristic as R increases or decreases away from 1.

4.2 $h - h^*$ Correlation

The next metric we consider is the correlation between h and h^*. While considering the percent error in h as a metric, we noted that greedy best-first search has run time linear in the solution length of the optimal solution if the nodes are in $h^*(n)$ order. One way to quantify this observation is to measure the correlation between the two values. We will do this in three different ways.

The most well known correlation coefficient is Pearson's correlation coefficient r, which 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 best-first search. For each domain, we calculated Pearson's correlation coefficient between $h^*(n)$ and $h(n)$, and the results are in the second column of Table 4.

Another reasonable way to measure the heuristic correlation is to use rank correlation. Rank correlation measures how well one permutation (or order) respects another permutation (or order). In the context of search, we can use this to ask how similar the order one gets by putting nodes in h order is to the order one gets by putting nodes in h^* order. Rank correlation coefficients are useful because they are less sensitive to outliers, and are able to detect relationships that are not linear.

Spearman's rank correlation coefficient (ρ) is the best known rank correlation coefficient. ρ is Pearson's r between the ranked variables. This means that the smallest of N heuristic values is mapped to 0, the largest of the n heuristic values is mapped to N. This is done for both h and h^*, at which point we simply calculate Person's r using the rankings. In the context of greedy best-first 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 best-first 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 best-first 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 4.

A more natural metric for measuring this relationship can be achieved by using Kendall's τ (1938). Kendall's τ is another rank correlation coefficient, but it measures the amount of concordance between the two rankings. Concordance is having the rankings for two elements agree. In the context of greedy best-first search, a concordant pair is a pair of nodes such that $h(n_1) > h(n_2)$ and $h^*(n_1) > h^*(n_2)$ or $h(n_1) < h(n_2)$ and $h^*(n_1) < h^*(n_2)$. Kendall's τ is the proportion of pairwise comparisons that are concordant. If h puts nodes in h^* order, all pairwise comparisons will be concordant, and Kendall's τ will be 1. If h puts nodes in reverse h^* order, all comparisons will be discordant, and Kendall's τ will be -1. If sorting nodes on h puts nodes in a random order, we expect that half of the comparisons will be concordant and half of the comparisons will be discordant.

Kendall's τ can also be understood in the context of bubble sort. The Kendall τ distance is the number of swaps that a bubble sort would do in order to change one list into the other. In this case, it is the number of swaps that a bubble sort would do rearranging a list of nodes sorted on h so the list is sorted on h^*. Kendall's τ is calculated by normalizing the Kendall τ distance, which is done by dividing by $N(N-1)/2$. [1]

Since τ and ρ are both rank correlation coefficients, they are related, but we argue that τ is the more natural statistic. Consider this question: given an open list containing n nodes, how likely is it that the node with the smallest h^* will be at the front of the open list, given that the nodes are ordered on h? We can use τ to predict that the node will, on average, be in the middle of the list if h and h^* are completely unrelated, and closer to the front of the open list the stronger the $\tau(h, h^*)$ correlation is. The reason is that if we assume that the nodes on the open list are a random selection of nodes, τ tells us how often a random comparison is correct. We can use therefore τ to predict how far back the node with the minimum h^* is. ρ has no such natural interpretation, making τ the more natural statistic. It is worth nothing that τ and ρ are generally related to one another, in that one can be used to predict the other (Gibbons, 1985). This relationship means that in practice, it is generally possible to use either metric.

Returning to Table 4, the results lead us to reject the correlation between h and h^* as a metric for predicting how well greedy best-first search will work. For all three correlation coefficients, there are examples of domains where greedy best-first search fails with high $h(n)$-$h^*(n)$ correlations, and examples of domains where greedy best-first search works well with poor $h(n)$-$h^*(n)$ correlations. For example, in TopSpin(3), we have a Kendall's τ of .42, but this is lower than the τ for Inverse Tiles and both City Navigation problems we consider.

4.3 $h - d^*$ Correlation

The strategy of greedy best-first search is to discover a goal quickly by expanding nodes with small $h(n)$ values. If nodes with small $h(n)$ are far away from a goal it is reasonable to believe greedy best-first search would perform poorly. We will denote by $d^*(n)$ the count of edges between a node n and the nearest goal, where distance is not measured by summing the cost of the edges in the path, but rather by counting the edges in the path.

$d^*(n)$ is equivalent to $h^*(n)$ if we modify the graph so that all edges cost 1. Looking at the plot of $h(n)$ vs $h^*(n)$ in the left half of Figure 10, 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. We denote the distance to the nearest goal in terms

1. Malte Helmert has noted (personal communication) that Kendall's τ, as described, is not an ideal metric to use for sequences that contain ties. For integer-valued heuristics, especially, ties may be very common. One way to account for ties in the rankings to use Kendall's τ-b statistic (Kendall & Gibbons, 1990) instead of τ (also known as τ-a). Kendall's τ-b accounts for ties in the rankings.

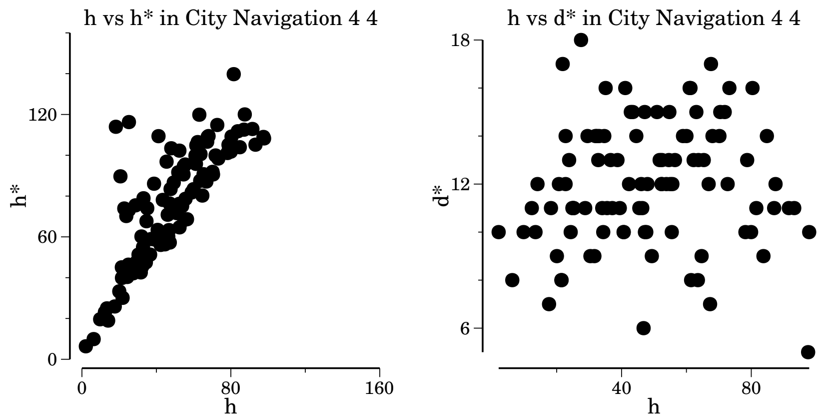

Figure 10: Plot of $h(n)$ vs $h^*(n)$, and $h(n)$ vs $d^*(n)$ for City Navigation 4 4

	Domain	$h(n)$-$d^*(n)$ Correlation (Pearson)	$h(n)$-$d^*(n)$ Correlation (Spearman)	$h(n)$-$d^*(n)$ Correlation (Kendall)
Greedy Works	Towers of Hanoi	0.9652	0.9433	0.8306
	Grid	0.9967	0.9958	0.9527
	Pancake	0.9621	0.9593	0.9198
	Dynamic Robot	0.9998	0.9983	0.9869
	Unit Tiles	0.7064	0.7065	0.5505
	TopSpin(3)	0.5855	0.4598	0.4158
Greedy Fails	TopSpin(4)	0.2827	0.3196	0.2736
	Inverse Tiles	0.5281	0.5173	0.3752
	City Nav 3 3	0.0246	-0.0338	-0.0267
	City Nav 4 4	0.0853	0.1581	0.1192

Table 5: Correlation between $h(n)$ and $d^*(n)$

of the number of edges in the state space graph as $d^*(n)$. The right half of Figure 10 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 best-first search works so poorly for this domain, despite the fact that $h(n)$ and $h^*(n)$ are so closely related.

If nodes with small $h(n)$ values are also likely to have small $d^*(n)$ values (and these nodes are therefore close to a goal, in terms of expansions away) expanding nodes with small $h(n)$ values 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 thus 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.

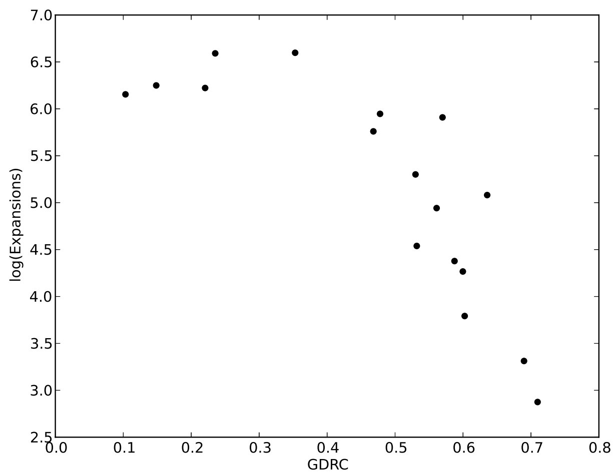

Figure 11: Average log of expansions done by greedy best-first search with different heuristics, plotted according to their GDRC.

For each domain, we quantify this concept by calculating Pearson's correlation coefficient, Spearman's rank correlation coefficient, and Kendall's τ between $d^*(n)$ and $h(n)$. Looking at Table 5, we can see that, using both Kendall's τ and Pearson's r, we are finally able to separate the domains on which greedy best-first search performs well from the domains on which greedy best-first search performs poorly. For Kendall's τ, we can draw a line at approximately 0.4 that can be used to separate the domains where greedy best-first search works well and the domains where greedy best-first search works poorly. Likewise, for Pearson's r, we can draw a line at approximately .55. We will call this type of metric the Goal Distance Rank Correlation (GDRC) and, unless otherwise noted, compute it using Kendall's τ.

The correlation between $d^*(n)$ and $h(n)$ connects to our three observations, although the connection is not a mathematical necessity (as counterexamples can be constructed). Note that a heuristic that obeys Observation 1 will produce paths where h monotonically decreases to a goal. Consider the nodes along a path to a goal. By hypothesis, h will monotonically decrease along this path. Now, consider one of the nodes at the goal end of the path. Since h monotonically decreases along the path, the nodes at the goal end of the path have a low h, and because they are near the end of the path, they also have a low d^* value. While little else can be said about the nodes in general, this restriction improves the heuristic's GDRC compared to a situation in which the nodes with low d^* are allowed to have high h values. A similar argument can be used to show how following Observation 2 helps to produce a heuristic with a high GDRC.

Observation 3 discusses nodes in a local minimum, and the difference in h between nodes in the local minimum, and nodes that are on the edge of the local minimum. If we assume that in order to escape a local minimum one must go through one of the nodes on the edge of the local minimum, then we know that the nodes in the local minimum must have a higher d^* than the nodes on the edge of the local minimum, but we also know that their h is lower (because the node is in the local minimum). This means that the h ranking incorrectly orders all nodes in the local minimum as compared to the nodes on the edge of the local minimum, a clear problem for producing a high GDRC. If the nodes in the local minimum have a low d^* because they can get to the goal through a very high h node, the relationship between this observation and GDRC is weaker. Consider personal transportation as an example. An action such as, "call a taxi" might result in reaching states very near the goal in one step, but at very high cost. If the heuristic recognizes the cost of this action, the node will correctly have a high h, but be very close to the goal as measured with d because of the call a taxi path. While such a situation clearly causes problems for domains attempting to follow Observation 3, we believe that domains with this kind of attribute are, in practice, quite uncommon. For example, none of our example domains exhibit this trait.

4.4 Comparing Heuristics

Because it is a quantitative metric, GDRC can be used to compare different heuristics for the same domain. To test its effectiveness, we ran experiments on the Towers of Hanoi problem using 17 different disjoint and non-disjoint pattern databases. We considered pattern databases with between 3 and 8 disks, as well as a selection of pairings of the PDBs where the total number of disks is less than or equal to 12. For each pattern database, we calculated the GDRC for the heuristic produced by the PDB. In Figure 11 we plot, for each PDB, the GDRC of the PDB on the X axis and the average of the log of the number of expansions required by greedy best-first search to solve 51 random 12-disk Towers of Hanoi problems on the Y axis. As we can see from the figure, when the GDRC is below roughly 0.4, greedy best-first search performs very poorly, but as the GDRC increases, the average number of expansions done by greedy best-first search decreases. This suggests that it is possible to use GDRC to directly compare heuristics against one another.

We can see similar behavior in a different domain in the left part of Figure 12. Each dot represents one of the 462 possible disjoint 5/6 pattern databases (one 6 tile PDB and one 5 tile PDB that are disjoint) for the 3×4 sliding tile puzzle with inverse costs. On the Y axis is the log of the average expansions required to solve 100 random instances. On the X axis is the GDRC. Since we are using a non-unit problem, h^* and d^* are not the same, so we can also calculate the correlation between h and h^*. In the right part of Figure 12, this correlation is on the X axis.

As we can see, GDRC and the rank correlation between h and h^* can both yield useful information about how well greedy best-first search is likely to work.

For the domains we have tested, the correlation between h and d^* neatly predicts when greedy best-first search performs worse than Weighted A* (or A*). It is not perfect, however. 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 such a heuristic could 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 best-first 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

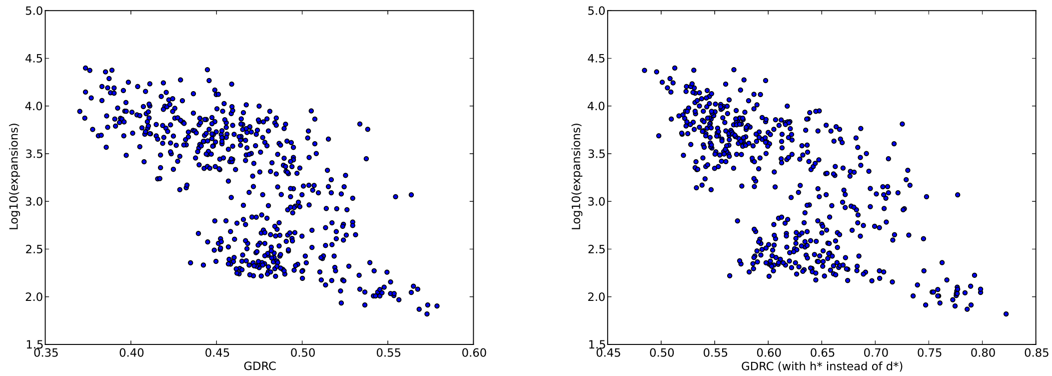

Figure 12: Average log of expansions done by greedy best-first search with each of the possible 462 5/6 disjoint PDB heuristics, plotted against GDRC (left) and the correlation between $h(n)$ and $h^*(n)$

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

or not greedy best-first 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 similar to 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 6 shows the various correlations and percent error in $h(n)$ for City Navigation 5 5. Figure 13 shows that as we increase the weight, despite the very weak correlation between $h(n)$ and $d^*(n)$, there is no catastrophe: greedy best-first 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.

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 best-first search, which leads us to the conclusion that in order for greedy best-first 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 remaining search distance (small $d^*(n)$) or close to the goal measured in terms of remaining cost (small $h^*(n)$). We

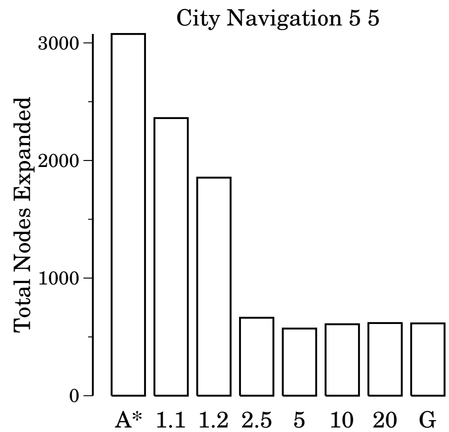

Figure 13: Expansions done by A*, Weighted A*, and greedy best-first search on City Navigation 5 5

have seen empirically that as the two correlations break down, the $d^*(n)$ correlation allows greedy best-first search to survive longer: in tested domains where the $d^*(n)$-$h(n)$ is above .58, greedy best-first search does well, whereas we have seen domains where the $h^*(n)$-$h(n)$ correlation is as high as .70 (or .75, depending on which correlation metric is being used) where greedy best-first search performs poorly.

The importance of the correlation between $h(n)$ and $d^*(n)$ reflects the importance of node ordering for greedy best-first search. In optimal search, the search cannot terminate when a solution is found, but rather when the solution is known to 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 best-first 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 the percent error in h. It is common for researchers to say that A*'s heuristic "guides" the search, but our discussion reveals why this language should be reserved for suboptimal search.

Some heuristics are able to satisfy both the needs of A* and greedy best-first search simultaneously. For example, the dynamic robot navigation heuristic works extremely well for both A* and greedy best-first search, because it is both big, and therefore good for A*, and good at differentiating nodes that are near the goal from those far away from the goal, helping greedy best-first search.

5. Building a Heuristic by Searching on GDRC

As shown by Haslum, Botea, Helmert, Bonet, and Koenig (2007), given a metric for assessing the quality of a heuristic, we can use that metric to automatically construct effective abstraction-based heuristics simply by searching the space of abstractions. In many domains, a heuristic can be constructed by initially abstracting everything, and slowly refining the abstraction to construct a heuristic. While Haslum et al. (2007) were concerned with optimal search and hence use pruning power to evaluate heuristics, our focus on greedy best-first search suggests that GDRC might serve as a useful metric. For example, in the TopSpin problem, we begin with a heuristic that abstracts all disks. We then consider all PDBs that can be devised by abstracting everything except one disk, and measure the GDRC of each pattern database. The GDRC can be effectively estimated by doing a breadth-first search backwards from the goal (we used 10,000 nodes for a 12 disk problem) to establish d^* values for nodes, and the h value can be looked up in the pattern database. We then sample 10% of the nodes generated in this way, and used the sample to calculate an estimate of Kendall's τ. While we elected to sample 10%, of the nodes, a sample of any size can be taken provided the confidence interval is sufficiently small to tell which τ is better. Last, we take the PDB with the highest value as the incumbent PDB. This process repeats until either all PDB's have a worse GDRC than the previous best PDB, or until the PDB has reached the desired size. The reason we allow the algorithm to possibly terminate early is to cover the case of GDRC decreasing with larger PDB's. If increasing the size of the PDB decreases GDRC, it is likely that further increasing the size of the PDB will degrade the GDRC even more, so we elect to terminate. The full algorithm is detailed in Algorithm 1. While this simple hill-climbing search appears effective, a more sophisticated search strategy could certainly be employed instead.

5.1 TopSpin

Algorithm 1 Hill Climbing PDB Builder

```
AllTokens = {Tokens in problem that can be abstracted}
RemainingTokens = AllTokens
BestPDB = build PDB by abstracting AllTokens
BestTau = 0
function tryPDB(tokens)
    pdb = build PDB by abstracting AllTokens \ tokens
    allNodes = nodes discovered by breadth first search backwards from the goal state(s)
    sample = randomly sample 10% of the nodes from allNodes
    return calcTau(sample, pdb)
while BestPDB.size < Max Allowed Size do
    LocalBestPDB, LocalBestTau, LocalBestToken = (None, BestTau, None)
    for all CurrentToken ∈ RemainingTokens do
        CurrentTau, CurrentPDB = tryPDB(RefinedTokens ∪ {token})
        if CurrentTau > LocalBestTau then
            set local best variables to current
    if LocalBestPDB ≠ None then
        set best variables to local best variables
        RemainingTokens = RemainingTokens \ LocalBestTokens
    else
        Break
return BestPDB
```

PDB	Greedy Exp	A* Exp	Avg. Value
Contiguous	411.19	10,607.45	52.35
Big Operators	961.11	411.27	94.37
Random	2,386.81	26,017.25	47.99

Table 7: Expansions to solve TopSpin problem with the stripe cost function using different PDBs

When used to generate unit-cost TopSpin pattern databases, hill-climbing on GDRC always produced PDBs where the abstracted disks were all connected to one another, and the refined disks were also all connected to one another. This prevents the abstraction from creating regions where $h = 0$, but where the goal is nowhere near the $h = 0$ nodes, per Observation 2.

With unit-cost TopSpin problems, abstractions where all of the disks are connected to one another work well for both greedy best-first search and A*. If we change the cost function such that moving an even disk costs 1 and moving an odd disk costs 10, we get the stripe cost function, so called because the costs are striped across the problem. The most effective PDBs for A* are those that keep as many odd disks as possible, because moving the odd disks is much more expensive than moving an even disk. If we use such a "big operator" pattern database for greedy best-first search, the algorithm will align the high cost odd disks, but will have great difficulty escaping from the resulting local minimum. If we use hill climbing on GDRC to build the heuristic, we end up with a "contiguous" heuristic that keeps the abstracted and the refined disks connected to one another. Table 7 provides results of how the various pattern databases did solving a suite of instances. We can see the importance of creating a good pattern database when we consider the Random row in the table, which contains the average number of expansions from 20 different randomly selected 6 disk pattern databases.

5.2 Towers of Hanoi

We can already infer from Figure 11 that, if we greedily select the PDB with the best τ from a collection of PDBs, we would select the best one. But it is certainly also possible to use a hill-climbing search to incrementally construct a PDB. When creating a PDB heuristic for the Towers of Hanoi, one maps the full size problem onto an abstracted version of the problem by removing some of the disks in the larger problem, and re-indexing the remaining disks so they map to the disks in the smaller problem. With this technique, a critical component in terms of performance is which disks are abstracted.

We define a mapping as a selection of disks to abstract. In our example, we once again consider a 12 disk problem using an 8 disk PDB, so we must select 4 of the 12 total disks to abstract. In Figure 14 the "+" glyphs each represent a randomly selected abstraction, and the heuristic it produced. As we can see, some abstractions produced extremely poor quality heuristics as measured by GDRC and by the average number of expansions done by greedy best-first search solving problems using that heuristic. Other heuristics fared significantly better both in terms of GDRC and average expansions by greedy best-first

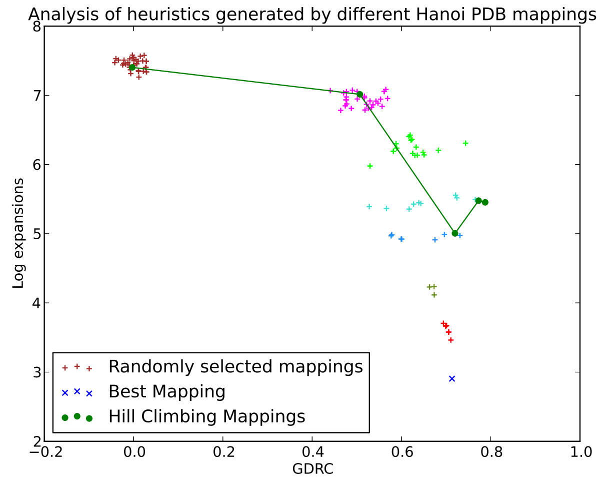

Figure 14: Expansions using different Towers of Hanoi PDB abstractions.

search. If we examine the plot in Figure 14 we can see that there are several clusters of heuristics. The heuristics with a GDRC of about 0 are clustered together, all requiring greedy best-first search to expand between 10^7 and 10^8 nodes. These heuristics are the worst heuristics, because they largest disk is abstracted. The next cluster of heuristics have a GDRC of about 0.4 to 0.5 and require greedy best-first search to expand between $10^{6.5}$ and 10^7 nodes. These are the heuristics where the largest disk is not abstracted, but the second largest disk is abstracted. These abstractions also produce very poor quality heuristics, but the heuristics represent a significant improvement over the heuristics where the largest disk is abstracted. Each color in Figure 14 represents a mapping with a different largest abstracted disk, with the blue "X" glyph representing the best pattern database where only the smallest disks are abstracted. As we can see in this plot, there is a definite overall trend where mappings that product heuristics with a higher GDRC tend to fare better overall in terms of average total expansions used by greedy best-first search.

The hill climbing algorithm selected the heuristic that contained disks 0, 1, 2, 4, 5, 6, 7, and 8 (skipping the 3 disk). An example of how the hill climbing algorithm selected this heuristic can be seen in the green circles and line in Figure 14 when starting from an abstraction that abstracted the largest disk. The hill climbing algorithm climbed a hill leading to a reasonable, albeit not the most effective, heuristic. Despite failing to find the optimal heuristic, the selected heuristic is quite reasonable nonetheless, falling between the 86^{th} and the 75^{th} percentile overall, a significant improvement for an automated approach.

5.3 City Navigation

In addition to building pattern database heuristics using hill climbing on GDRC, it is also possible to build a portal-style heuristic by hill climbing on GDRC. Using the city navigation domain, we defined a portal heuristic (Goldenberg, Felner, Sturtevant, & Schaeffer, 2010) by selecting a number of nodes to be portal nodes (we used the same number of nodes as

Heuristic	Average GDRC	Average greedy best-first search expansions
Random Portals	0.44	2200
Nexus Portals	0.76	488
Hill Climbed Portals	0.60	1117

Table 8: Expansions and GDRC using different ways to select portal nodes

cities, 150), and calculating the true distance from every node to the closest portal node. The heuristic for two nodes is the true distance between the portals associated with each node minus the distance of each node to its own portal. In the event that this quantity is negative, 0 is used. This heuristic is highly accurate across long distances because it uses the true distance between the portals, but it is obviously less accurate when comparing two nodes that share the same portal. When constructing portal heuristics, the critical difference between an effective portal heuristic and a poor quality portal heuristic is the selection of which nodes are portals. We allowed our algorithm to automate this process by hill climbing on GDRC. The algorithm is initialized with a random array of nodes as the portals. At each step, the algorithm iterates through the indexes in its array of portals, considering moving the location that is currently serving as a portal to a different location. In our implementation, we considered two times the number of cities, or 300 different random places. We then assessed the GRDC of the new heuristic using a sample of 100,000 randomly selected pairs of places. If moving the city to a new location improved GDRC, we kept the portal array with the new place, otherwise, we discarded the change as its GDRC is inferior to that of the incumbent. When we reach the end of the array, we restart at the beginning. If we reach a point where we are at the same position in the array, and all other aspects of the array remain unchanged since the last time we modified that index, the algorithm terminates, returning the array of portals for use in a heuristic.

Results from this experiment are shown in Table 8. The average GDRC is the GDRC that one obtains by selecting 100,000 random pairs of start and end nodes and calculating GDRC using those nodes. The average greedy best-first search expansions is the average number of expansions needed to solve a City Navigation problem with a random start and goal.

We considered three different methods for selecting portal nodes. The first was to completely randomize the selection of portal nodes, which unsurprisingly resulted in the lowest GRDC and the highest number of expansions. The most successful method for selecting portal nodes was to identify the nexus nodes, and use those nodes as the portals. Unsurprisingly, this method led to the highest GDRC, and the fewest number of expansions. This result further demonstrates the usefulness of GDRC in identifying a quality heuristic for greedy best-first search. Last, our automatic algorithm for finding portal nodes performed significantly better than random, while still trailing the hand-selected portals. We believe that a better search strategy may be able to better capture the potential performance gain offered by high GDRC heuristics.

5.4 Sliding Tile Puzzle

We can also compare the GDRC-generated PDBs to instance-specific PDBs for the sliding tile puzzle (Holte et al., 2005). On this domain, in order to get an accurate estimate of τ, we had to increase the number of nodes expanded going backwards from 10,000 to 1,500,000. Following the hill climbing procedure, the algorithm selected a pattern database that tracked the 1, 3, 4, 7, 8, and 11 tiles. The results of using this PDB are shown in Table 3. While this abstraction is not as strong as the outer L abstraction, it is the fourth best PDB for minimizing the average number of expansions done by greedy best-first search out of the 462 possible 6-tile pattern databases. The automatically constructed PDB is two orders of magnitude faster than the number of expansions one would expect to do using an average 6-tile PDB, and three orders of magnitude faster than the worst 6-tile PDB for greedy best-first search. The GDRC-generated PDB works substantially better for greedy best-first search then the state-of-the-art instance-specific PDBs, requiring about one twentieth of the expansions. One additional advantage that the GDRC-generated PDB has over instance-specific PDBs is the fact that GDRC produces a single PDB, unlike instance specific PDBs, which produce a new PDB for every problem.

In summary, these results show that GDRC is useful for predicting the relative quality of heuristics for greedy best-first search. They also showed that it is possible to leverage this quantitative metric to automatically construct a heuristic for greedy best-first search, and that the automatically created heuristics are extraordinarily effective for greedy best-first search.

6. Related Work

As a metric, GDRC predicts that heuristics that have a high rank correlation with d^* will work well. In general, the objective of h is to approximate h^*, not d^*, so one alternative way to find a quality heuristic is to leverage this fact and try to construct a heuristic directly that mimics d^*, generally referred to as d. Indeed, this approach is generally quite successful (as opposed to relying exclusively on h), handily outperforming h in many situations (Wilt & Ruml, 2014).

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 limitations. 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 on the amount of error present in the heuristic, where error is measured as relative deviation from $h^*(n)$. For A*, this criterion makes a certain amount of sense, but for greedy best-first search, we have seen that relative deviation from $h^*(n)$ cannot be used to predict when greedy best-first search will perform poorly. Gaschnig points out that increasing the weight ad infinitium may decrease performance, which is precisely the phenomenon we documented in Section 2.

Chenoweth and Davis (1991) show that if the heuristic is "rapidly growing with logarithmic cluster", a greedy best-first 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

$g(x) = x$. We are not aware of any heuristics that have been proven to be rapidly growing with logarithmic cluster.

A number of works consider the question of predicting search algorithm performance (Korf et al., 2001; Pearl, 1984; Helmert & Röger, 2008), although the subject attracting by far the most attention is determining how many nodes will be expanded by an optimal search algorithm. As we saw in Section 3, the behavior of optional search does not in general predict the behavior of GBRS. Lelis, Zilles, and Holte (2011) did an empirical analysis of suboptimal search algorithms, predicting the number of nodes that would be expanded by Weighted IDA*, but it is not clear if those methods can predict greedy best-first search behavior, and thus tell us if increasing the weight too far can be detrimental.

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 our work.

Hoffmann (2005) discusses why the FF heuristic (Hoffmann & Nebel, 2001) is an effective way to solve many planning benchmarks when used in conjunction with enforced hill climbing. The paper shows that in many benchmark problems, the heuristic has small bounded-size plateaus, implying that the breadth-first search part of the enforced hill climbing algorithm is bounded, which means that those problems can be solved quickly, sometimes in linear time. Although enforced hill climbing is a kind of greedy best-first search, its behaviour is very different from greedy best-first search when a promising path turns into a local minimum. Greedy best-first search considers nodes from all over the search space, possibly allowing very disparate nodes to compete with one another for expansion. Enforced hill climbing limits consideration to nodes that are near the local minimum (with nearness measured in edge count), which means that the algorithm only cares about how the heuristic performs in a small local region of the space. Hoffmann (2011) extends this concept, describing a process for automatically proving that a domain will have small local minima.

Xu, Fern, and Yoon (2009) discuss constructing heuristics for a suboptimal heuristic search, but the algorithm they consider is a beam search. Beam searches inadmissibly prune nodes to save space and time, so their function is ultimately being used not to rank nodes, but to make a decision as to whether or not to keep any one node. The function that Xu et al. create can be used to rank nodes, but the input function requires a variety of features of the state to function, and is created by using training data from trial search runs. Our approach of creating a heuristic by hill-climbing on GDRC does not require training instances, nor does it require any information about the states themselves. Hill-climbing on GDRC does, however, have the limitation that the automatic generation of heuristics only works when an appropriate search space can be defined, as with abstraction-based heuristics.

7. Conclusion

Suboptimal heuristic searches rely heavily on the heuristic node evaluation function. We first showed that greedy best-first search can sometimes perform worse than A*, and that although in many domains there is a general trend where a larger weight on the heuristics 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 best-first search has no bounds on performance, and given a poor heuristic, greedy best-first search could very well expand the entire state space, or never terminate if the state space is infinite. Our work shows that poor performance is not just a theoretical curiosity, but that this behavior can occur in practice.

We then considered characteristics of effective heuristics for greedy best-first search. We showed several examples in which the conventional guidelines for building heuristics for A* can actually harm the performance of greedy best-first search. We used this experience to develop alternative observations and desiderata for heuristics for use with greedy best-first search. The first is that from every node, there should be a path to a goal that only decreases in h. The second, an important special case of the first, is that nodes with $h = 0$ should be connected to a goal via nodes with $h = 0$. The third observation is that nodes that require including high h nodes in the solution should themselves have as high an h value as possible.

We then showed that the domains where greedy best-first search is effective share a common trait of the heuristic function: the true distance from a node to a goal, defined as $d^*(n)$, correlates well 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 which algorithm to use.

Finally, we showed that goal distance rank correlation (GDRC) can be used to compare different heuristics for greedy best-first search, and demonstrated how it can be used to automatically construct effective abstraction heuristics for greedy best-first search.

Recent work has shown that search algorithms explicitly designed for the suboptimal setting can outperform methods like weighted A*, which is a simple unprincipled derivative of an optimal search (Thayer & Ruml, 2011; Thayer, Benton, & Helmert, 2012; Stern, Puzis, & Felner, 2011). Our results indicate that the same holds true for heuristic functions as well: suboptimal search deserves its own specialized methods. Given the importance of suboptimal methods in solving large problems quickly, we hope that this investigation spurs further analysis of suboptimal search algorithms and the heuristic functions they rely on.

8. Acknowledgments

We gratefully acknowledge support from NSF (award 1150068). Preliminary expositions of these results were published by Wilt and Ruml (2012, 2015).

References

Burns, E. A., Hatem, M., Leighton, M. J., & Ruml, W. (2012). Implementing fast heuristic search code. In *Proceedings of the Fifth Symposium on Combinatorial Search*.

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

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

Felner, A., Korf, R. E., Meshulam, R., & Holte, R. C. (2007). Compressed pattern databases. *Journal of Artificial Intelligence Research (JAIR)*, *30*, 213–247.

Felner, A., Zahavi, U., Holte, R., Schaeffer, J., Sturtevant, N. R., & Zhang, Z. (2011). Inconsistent heuristics in theory and practice. *Artificial Intelligence*, *175*(9-10), 1570–1603.

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*, pp. 434–441.

Gibbons, J. D. (1985). *Nonparametric Statistical Inference*. Marcel Decker, Inc.

Goldenberg, M., Felner, A., Sturtevant, N., & Schaeffer, J. (2010). Portal-based true-distance heuristics for path finding. In *Proceedings of the Third Symposium on Combinatorial Search*.

Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, *SSC-4*(2), 100–107.

Haslum, P., Botea, A., Helmert, M., Bonet, B., & Koenig, S. (2007). Domain-independent construction of pattern database heuristics for cost-optimal planning. In *Proceedings of AAAI-07*, pp. 1007–1012.

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*.

Helmert, M., & Röger, G. (2008). How good is almost perfect?. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI-2008)*, pp. 944–949.

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

Hoffmann, J. (2011). Analyzing search topology without running any search: On the connection between causal graphs and h^+. *Journal of Artificial Intelligence Research*, *41*, 155–229.

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

Holte, R., Grajkowskic, J., & Tanner, B. (2005). Hierachical heuristic search revisitied. In *Symposium on Abstracton Reformulation and Approximation*, pp. 121–133.

Kendall, M. G. (1938). A new measure of rank correlation. *Biometrika*, *30*(1/2), 81–93.

Kendall, M., & Gibbons, J. D. (1990). *Rank Correlation Methods* (Fifth edition). Edward Arnold.

Korf, R., & Felner, A. (2002). Disjoint pattern database heuristics. *Artificial Intelligence*, *134*, 9–22.

Korf, R. E. (1987). Planning as search: A quantitative approach. *Artificial Intelligence*, *33*(1), 65–88.

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

Korf, R. E. (1997). Finding optimal solutions to Rubik's cube using pattern databases. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, AAAI'97, pp. 700–705. AAAI Press.

Korf, R. E. (2007). Analyzing the performance of pattern database heuristics. In *Proceedings of the 22nd National Conference on Artificial Intelligence*, AAAI'07, pp. 1164–1170. AAAI Press.

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

Korf, R. E., & Taylor, L. A. (1996). Finding optimal solutions to the twenty-four puzzle. In *AAAI, Vol. 2*, pp. 1202–1207.

Lelis, L., Zilles, S., & Holte, R. C. (2011). Improved prediction of IDA*'s performance via epsilon-truncation. In *Proceedings of the Fourth Symposium on Combinatorial Search*.

Likhachev, M., Gordon, G., & 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., & Ferguson, D. (2009). Planning long dynamically feasible maneuvers for autonomous vehicles. *International Journal Robotic Research*, *28*(8), 933–945.

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

Nilsson, N. J. (1980). *Principles of Artificial Intelligence*. Tioga Publishing Co.

Parberry, I. (1995). A real-time algorithm for the (n^2-1)-puzzle. *Information Processing Letters*, *56*(1), 23–28.

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

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

Stern, R. T., Puzis, R., & Felner, A. (2011). Potential search: A bounded-cost search algorithm. In *Proceedings of the 21st International Conference on Automated Planning and Scheduling, ICAPS*.

Sussman, G. J. (1975). *A Computer Model of Skill Acquisition*. New York: New American Elsevier.

Thayer, J. T., Benton, J., & Helmert, M. (2012). Better parameter-free anytime search by minimizing time between solutions. In *Proceedings of the Fifth Annual Symposium on Combinatorial Search, SOCS 2012*.

Thayer, J. T., & Ruml, W. (2011). Bounded suboptimal search: A direct approach using inadmissible estimates. In *Proceedings of the Twenty Sixth International Joint Conference on Articial Intelligence (IJCAI-11)*, pp. 674–679.

Tukey, J. W. (1977). *Exploratory Data Analysis*. Addison-Wesley, Reading, MA.

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

Wilt, C., & Ruml, W. (2012). When does weighted A* fail?. In *Proceedings of the Fifth Symposium on Combinatorial Search*.

Wilt, C., & Ruml, W. (2014). Speedy versus greedy search. In *Proceedings of the Seventh Symposium on Combinatorial Search*.

Wilt, C., & Ruml, W. (2015). Building a heuristic for greedy search. In *Proceedings of the Eighth Symposium on Combinatorial Search*.

Xu, Y., Fern, A., & Yoon, S. (2009). Learning linear ranking functions for beam search with application to planning. *The Journal of Machine Learning Research*, *10*, 1571–1610.