Building a Heuristic for Greedy Search

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

Abstract

Suboptimal heuristic search algorithms such as greedy best-first search allow us to find solutions when constraints of either time, memory, or both prevent the application of optimal algorithms such as A*. Guidelines for building an effective heuristic for A* are well established in the literature, but we show that if those rules are applied for greedy best-first search, performance can actually degrade. Observing what went wrong for greedy best-first search leads us to a quantitative metric appropriate for greedy heuristics, called Goal Distance Rank Correlation (GDRC). We demonstrate that GDRC can be used to build effective heuristics for greedy best-first search automatically.

Introduction

The A* search algorithm finds optimal solutions with optimal efficiency (Dechter and Pearl 1988). For many problems, however, finding an optimal solution will consume too much time or too much memory, even with a heuristic with only constant error (Helmert and Röger 2007). For problems that are too difficult for A*, practitioners often consider Weighted A* (Pohl 1970) or greedy best-first search (GBFS) (Doran and Michie 1966). GBFS is a best-first search that expands nodes in h order, preferring nodes with small h, dropping duplicates by using a closed list. The heuristic plays a very large role in the effectiveness of these algorithms.

For A*, there are a number of well-documented techniques for constructing an effective heuristic. In this paper, we revisit these guidelines in the context of GBFS. We begin by showing that if one follows the established guidelines for creating a quality heuristic for A*, the results are decidedly mixed. We present several examples where following the A* wisdom for constructing a heuristic leads to slower results for greedy search.

Using these failures, we derive informal observations about what seems to help GBFS perform well. We then develop a quantitative metric that can be used to compare heuristics for use with GBFS. We demonstrate how it can be used to automatically construct an effective heuristic for GBFS by iteratively refining an abstraction. The resulting heuristics outperform those constructed using methods developed for optimal search algorithms such as A*.

This work provides both qualitative and quantitative ways to assess the quality of a heuristic for the purposes of GBFS. Because GBFS is one of the most popular and scalable heuristic search techniques, it is important to develop a better understanding of how to use it. Lessons from optimal search do not necessarily carry over to the suboptimal setting.

Building a Heuristic for A*

We begin by briefly reviewing the literature on constructing a heuristic for the A* search algorithm. For finding optimal solutions using A*, the first and most important 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 $h(n)$.

The most widespread rule for making a good heuristic for A* is: dominance is good (Pearl 1984). A heuristic h_1 is said to dominate h_2 if $\forall n : h_1(n) \geq h_2(n)$. This makes sense, because due to admissibility, larger values are closer to h^*. Furthermore, A* expands every node n it encounters where $f(n) < f(opt)$, so large h often reduces expansions. When it is difficult to prove dominance between two heuristics, they 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 and duplicate states, A* and the last iteration of Iterative Deepening A* (IDA*) expand the same number of nodes. Korf, Reid, and Edelkamp (2001) predict that the number of nodes IDA* will expand at cost c is:

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

The function $P(h)$ 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, Reid, and Edelkamp 2001). This quantity tends to decrease as h gets larger, depending on how the nodes in the space are distributed.

When considering pattern database (PDB) heuristics (Culberson and Schaeffer 1998), assuming the requirements

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 1: Average number of nodes expanded to solve 51 random 12 disk 4 peg Towers of Hanoi problems.

of A* and IDA* are the same also allows us to apply Korf's Conjecture (Korf 1997), which tells us that we can expect $M \times t = n$, where $M = \frac{m}{1+log(m)}$ where m is 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). This equation tells us that we should expect larger pattern databases to provide faster search. To summarize, bigger is better, both in terms of average heuristic value and pattern database size.

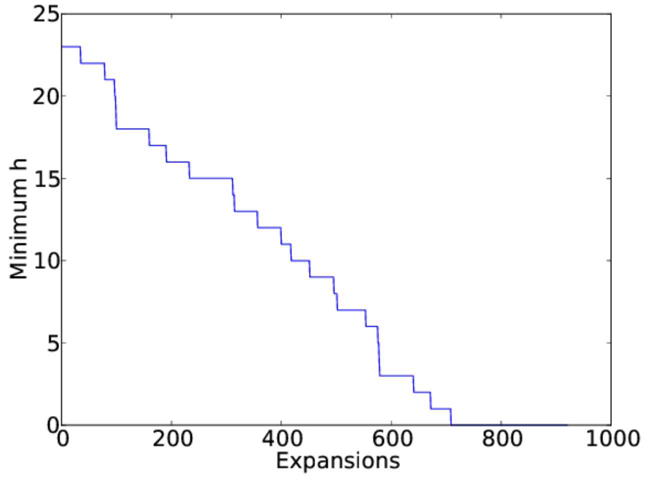
The Behavior of Greedy Best-First Search

As we shall see, these guidelines for A* heuristics are all very helpful when considering A*. What happens if we apply these same guidelines to greedy best-first search? We answer this question by considering the behavior of greedy search on three benchmark problems: the Towers of Hanoi, the TopSpin puzzle, and the sliding tile puzzle.

Towers of Hanoi Heuristics

The most successful heuristic for optimally solving 4-peg Towers of Hanoi problems is disjoint pattern databases (Korf and Felner 2002). Disjoint pattern databases boost the heuristic value by providing information about the disks on the top of the puzzle. For example, consider 12 disks, 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. The exact numbers are presented in the Unit rows of Table 1 (the other rows correspond to alternate cost metrics which are discussed later). 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 dominance relation also transfers to the KRE equation, meaning that if a heuristic h_1 dominates a different heuristic h_2, the KRE equation predicts that the expected expansions using h_1 will be less than or equal to the expected expansions using h_2.

With greedy search, however, faster search results when we do not use a disjoint pattern database, and instead only use the result of the 8 disk pattern database. The reason for the different behaviour of A* and GBFS 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.



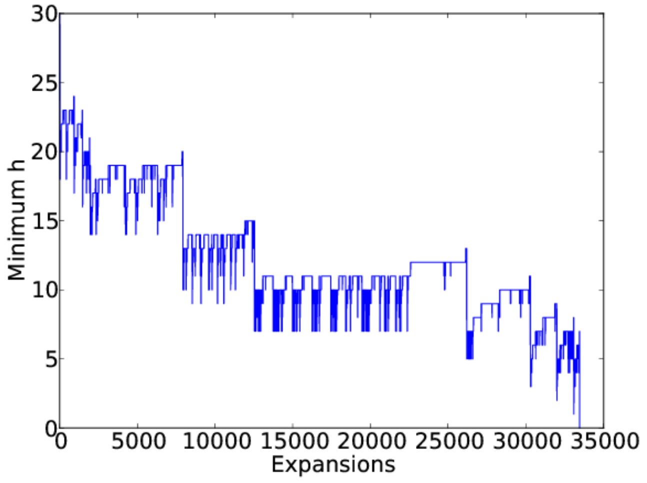


Figure 1: The h value of the expanded node as a function of the number of expansions. Top, a single PDB, bottom, two disjoint PDBs.

To see this, note that every combination of the bottom disks has an h value specified by the bottom PDB, and all possible arrangements of the disks on top will also share that same h value. Since the disks on top can be moved around independently of where the bottom disks are, 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, until 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 search simply considers possible configurations of the top disks until a goal has been found.

This phenomenon can be seen in the top pane of Figure 1, where the minimum h value of the open list monotonically decreases as the number of expansions the search has done increases. The heuristic created by the single pattern database creates an extremely effective gradient for the greedy search algorithm to follow for two reasons. First, there are no local minima at all, only the global minimum where the goal is. Following Hoffmann (2005) we define 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 bottom pane of Figure 1, 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 GBFS'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.

To summarize, the disjoint pattern database makes a gradient that is more difficult for greedy 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 brings us to our first observation.

Observation 1. *All else being equal, greedy 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.

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 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. GBFS uses the heuristic to specify 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 is compatible with 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.

These effects can be exacerbated if the cost of the disks on the top are increased relative to the costs 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 that, once again, the cost of moving a disk is linearly proportional to the disk's size, but the disks are stacked

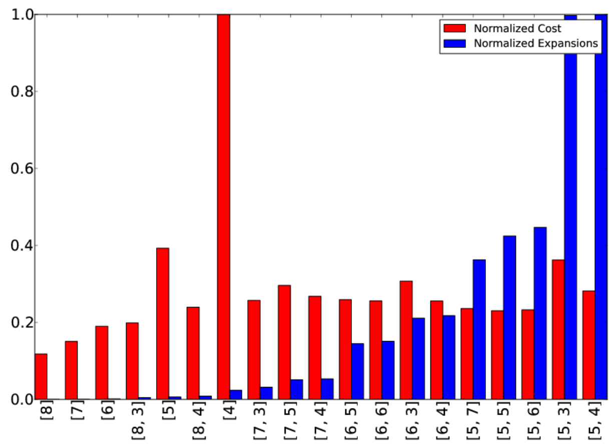


Figure 2: Normalized solution cost and expansions attained by GBFS using various PDBs on Towers of Hanoi.

in reverse, requiring 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 still expect that the number of expansions that greedy search will require will be lower when using only the bottom pattern database, and this is indeed the effect we observe in Table 1. However, if the top disks are heavier than the disks on the bottom, greedy search suffers even more with disjoint PDBs 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. If the situation is reversed, however, and the top pattern database uses only low cost operators, the top pattern database comprises a much smaller proportion of h. Since greedy search 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.

Solution Quality While not the central focus on this paper, it is important to consider solution quality because many users are concerned with both execution time as well as solution quality. Solution quality is simply the sum of the cost of all actions included in the final path. All else being equal, less expensive paths are more desirable than more expensive paths, although it is typically the case that the less expensive paths are also more computationally intensive to find. In Figure 2 we consider single pattern databases with only one pattern database with between 4 and 8 disks. We also consider disjoint pattern databases with two disjoint parts with at least five disks in the bottom PDB, and between 4 and 7 disks in the top PDB. As we can see, solution quality varies less than an order of magnitude, and if we disregard the single outlier, there is little variation in solution quality. We can also see that the best pattern databases for expansions all result in very high quality solutions.

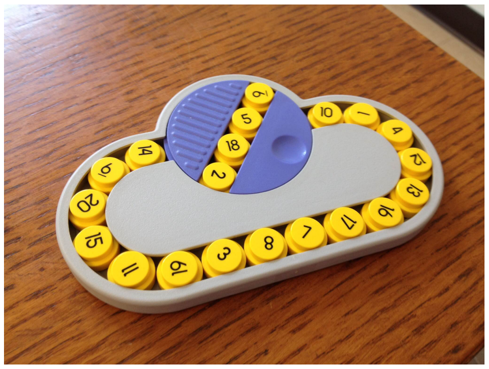


Figure 3: A TopSpin puzzle

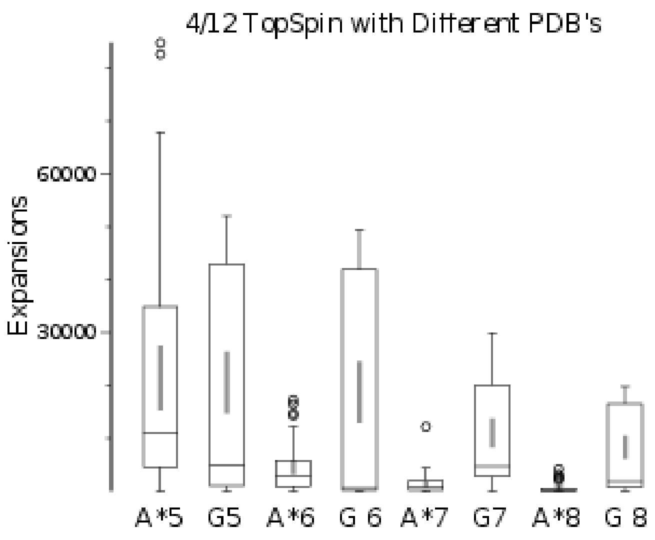


Figure 4: Total expansions done by GBFS on the TopSpin puzzle with different heuristics

TopSpin Heuristics

In the TopSpin puzzle, the objective is to sort a permutation by iteratively reversing a contiguous subsequence of fixed size. An example can be seen in Figure 3. We used a simplified problem with 12 disks and a turnstile that flipped 4 disks, because for larger puzzles, greedy search would sometimes run out of memory. We considered pattern databases that contained 5, 6, 7, and 8 of the 12 disks.

Korf's conjecture predicts that the larger pattern databases will be more useful for A* and indeed, this can be seen in Figure 4. Each box plot (Tukey 1977) is labeled with either A* or G (for GBFS), and a number, denoting the number of disks that the PDB tracks. Each box denotes the middle 50% of the data, with a horizontal line indicating the median. Circles denote outliers, whiskers represent the range of the non-outlier data, and thin gray stripes indicate 95% confidence intervals on the mean. 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

	A	A	3
A	A	A	7
8	9	10	11

	1	A	3
4	A	6	A
A	9	A	11

Figure 5: Different tile abstractions

greedy search, the gains are nowhere near as impressive as for A*.

The reason that greedy best-first search does not perform better when given a larger heuristic is that states with $h = 0$ may still be quite far from a goal. For example, consider the following TopSpin state where A denotes an abstracted disk:
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:
0 1 2 3 4 5 6 7 8 9 11 10

Moving a disk that is not abstracted will increase the heuristic, so this means that the TopSpin subgraph of only nodes with $h = 0$ is disconnected. Thus, when greedy search encounters a state with $h = 0$, it may turn out that the node is nowhere near the goal. If this is the case, greedy search will first expand all the $h = 0$ nodes connected to the first $h = 0$ node, 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 with only $h = 0$ 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 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 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.

Sliding Tiles Heuristics

The sliding tile puzzle is one of the most commonly used and best understood benchmark domains in heuristic search. Pattern database heuristics have been shown to be the strongest

[1]The average across all PDBs for A* is based upon the 429 PDBs which A* was able to use to solve all instances

Abstraction	Greedy Exp	A* Exp
Outer L (Figure 5 left)	258	1,251,260
Checker (Figure 5 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 τ Generated	427	1,197,789
Average 6 tile PDB [1]	17,641	1,596,041
Worst 6 tile PDB (for GBFS)	193,849	1,911,566

Figure 6: Performance of GBFS and A* using pattern databases generated with different abstractions. DNF denotes at least one instance requires more than 8GB to solve.

heuristics for this domain (Korf and Taylor 1996). We use the 11 puzzle (4x3) as a case study because the smaller size of this puzzle allows us to enumerate all pattern databases with 6 preserved tiles. The central challenge when constructing a pattern database for domains like the sliding tile puzzle is selecting a good abstraction.

The abstraction that keeps only the outer L, shown in the left part of Figure 5 is extremely effective for greedy search, because once greedy search has put all non-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. Compare this to what happens when greedy search is run on a checkerboard abstraction of the same size, as shown in the top right part of Figure 5. Once greedy 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 the first two rows of the table in Figure 6, where the average number of expansions required by A* is comparable with either abstraction, while the average number of expansions required by greedy search is larger by two orders of magnitude.

The sheer size of the PDB is not as important for GBFS as it is for A*. In Figure 6, we can see that as we weaken the pattern database by removing the 3 and 7 tiles, the number of expansions required increases only modestly for GBFS, while rendering some problems unsolvable for A* within 8 GB of memory. 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 search using certain kinds of pattern databases is the fact that the less useful pattern databases have nodes with $h = 0$ that are nowhere near the goal. This provides additional confirmation that Observation 2 matters; 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. A* is able to eliminate some of these states from consideration by considering the high g value that such states may have.

The checkerboard pattern database also helps to make clear another problem facing greedy 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 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 visible when considering instance specific pattern databases, a state of the art technique for constructing pattern databases for A* (Holte, Grajkowskic, and 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 can have the undesirable side effect of raising the h value of all nodes near the root, including nodes that are required to be included in a path to the goal. 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 the table in Figure 6. Unfortunately, despite its power with A*, if we apply this technique to greedy best-first search, the resulting pattern database is still not nearly as powerful as the outer L abstraction.

This observation contrasts with how Hoffmann (2005) discusses the "size" of a local minimum. Hoffmann considers the performance of the enforced hill climbing algorithm. This algorithm begins by hill climbing, always pursuing a node with a lower h value. When the algorithm cannot find a node with lower h, it then begins doing a breadth-first search, using the node with the lowest h as a root, until a node with a lower h is found, at which point hill climbing resumes. As such, the performance of this algorithm is dependent upon how far away the nearest node with a lower heuristic is from the origin point of the breadth-first search. Thus, for this algorithm, the important question is how many edges are between the origin point of the breadth-first search and the termination of the breadth-first search at a node with a lower h value.

Greedy best-first search considers all nodes on the open list when deciding which node to expand, and selects nodes with low h no matter where the they are in the search space. The breadth-first aspect of enforced hill-climbing causes the search to expand only nodes that are "near" the root of the breadth-first search (with distance defined using edge count, not edge weight), a bias that greedy best-first search lacks. Thus, for greedy best-first search, the question is how many nodes are actually in the local minimum, independent of where those nodes are with respect to the bottom of the local minimum that is the origin point of the breadth-first search that enforced hill-climbing does. While measuring how far the breadth-first search has to go to find a better node is helpful for enforced hill-climbing, this number is independent of how much work a greedy best-first search is going to have to do, because it is unrelated to the total number of nodes in the local minimum.

Quantifying Effectiveness in Greedy Heuristics

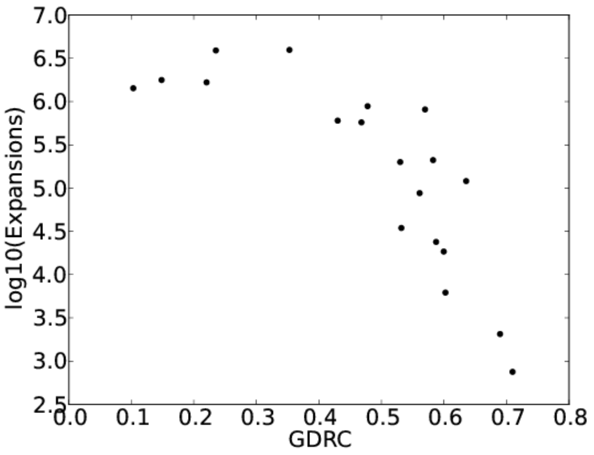
Our observations provide qualitative suggestions for what GBFS looks for in a heuristic, but quantitative metrics for assessing heuristic quality are also extremely helpful. It is important to note that from the perspective of GBFS, the magnitude of h is unimportant, and all that truly matters is node ordering using h. GBFS will be fastest if ordering nodes on h puts the nodes in order of $d^*(n)$ (the true distance-to-go, or the cost of going from n to a goal where all edges cost 1). One way to quantify this idea is to assess how well ordering nodes on h replicates the ordering on d^*. Using rank correlation, one can assess how well one ordering replicates another ordering. We call the rank correlation between h and d^* the Goal Distance Rank Correlation, or GDRC.

Spearman's ρ is a rank correlation metric that measures the correlation (Pearson's r) between the ranked variables. Another way to quantify rank correlation is Kendall's τ (Kendall 1938) which compares counts of concordant pairs and discordant pairs. In the context of greedy search, a concordant pair of nodes p_c is a pair of nodes such that $h(n_1) > h(n_2)$ and $d^*(n_1) > d^*(n_2)$ or $h(n_1) < h(n_2)$ and $d^*(n_1) < d^*(n_2)$. p_d is a pair of nodes that is not concordant, and n is the total number of nodes in the space. Kendall's τ is equal to $\frac{|p_c|-|p_d|}{0.5 \cdot n(n-1)}$. It is also possible to estimate the true value of Kendall's tau by sampling nodes.

In this paper we use Kendall's τ to measure GDRC, but it is worth noting that because both τ and ρ are measures of rank correlation, they are generally related to one another, in that one can be used to reliably predict the other (Gibbons 1985). This relationship means that in practice, it is generally possible to use either metric.

As h deviates from our observed requirements, we expect GDRC to decrease. For example, if a heuristic violates Observation 1 or Observation 2, solutions starting at a node with low h or $h = 0$ require at least one high h node. This means there are some nodes with low h or $h = 0$ that have higher d^* than some nodes with high h, which means there will be errors in the rank ordering. If a heuristic violates Observation 3, solutions starting at a node in a local minimum require high h nodes, which once again forces some nodes with higher h to have lower d^*, and forces some nodes with lower h to have higher d^*.

In order to assess the effectiveness of GDRC, we ran experiments on the Towers of Hanoi problem using 19 disjoint and non-disjoint pattern databases. We considered single pattern databases with between 4 and 8 disks, as well as all possible pairings of PDBs where the total number of disks



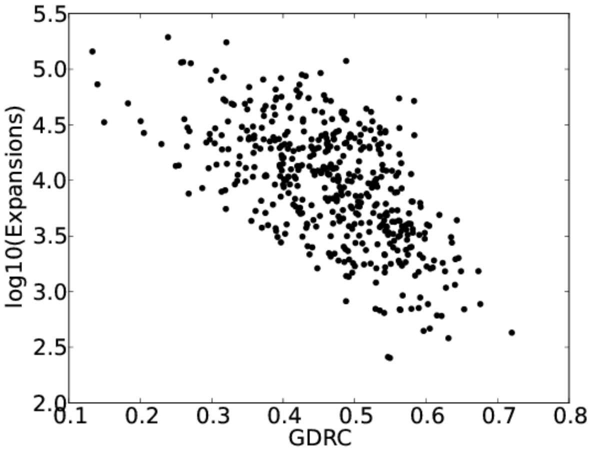


Figure 7: Average log of expansions with different heuristics, plotted against τ GDRC (top is Towers of Hanoi, bottom is 11-puzzle)

in all PDBs is less than or equal to 12, each constituent PDB contained between 3 and 8 disks, and the bottom PDB never contained fewer than 5 disks. Pattern databases where the bottom PDB contained fewer than 5 disks sometimes exceeded resource bounds. For each pattern database, we calculated the GDRC for the heuristic produced by the PDB. In the top part of Figure 7 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 on the Y axis. As we can see from the figure, when GDRC is below roughly 0.4, greedy search performs very poorly, but as GDRC increases, the average number of expansions done by GBFS decreases.

We also did experiments on all 462 distinct 6 tile PDBs for the 11 puzzle (Figure 7 bottom). Once again, for each PDB we estimated Kendall's τ, and calculated the average number of expansions GBFS required to solve the problem using that pattern database. As was the case with the Towers of Hanoi, we can see that as the GDRC increases, the average number of expansions required by GBFS decreases.

To assess the helpfulness of GDRC, we also compared evaluating heuristics using the dominance relationship and by using GDRC. Given a set of heuristics we begin by considering all pairs of heuristics, and attempt to pick which heuristic in each pair will lead to faster GBFS. When using GDRC to pick a heuristic, we simply select the heuristic with the larger GDRC, and this is either the correct decision (GBFS using the selected heuristic expands fewer nodes than GBFS using the alternate heuristic) or the incorrect decision. As a simple example, we could compare a single PDB with the bottom 8 disks ($\tau = 0.709$ for reverse square costs) against a disjoint PDB with the bottom 6 disks and the top 6 disks ($\tau = 0.430$ for reverse square costs). Since the PDB with the bottom 8 disks has a higher τ, we select this heuristic, since we expect it to be a superior heuristic, which turns out to be true, since using the single 8 disk PDB with reverse square costs requires an average of 771 expansions, while using the disjoint 6/6 PDB requires an average of approximately 646,000 expansions. When we select between heuristics using GDRC, we make the correct decision on 159/171 pairings of heuristics on the Towers of Hanoi with square costs, and we make the correct decision 157/171 pairings when using the reverse square costs.

We can also perform this same evaluation using the dominance relationship. For example, keeping with the reverse square example from before, an 8 disk PDB dominates a 7 disk PDB, so if selecting between a 7 and an 8 disk PDB, we would expect the 8 disk PDB to be superior, which is true (average of 771 expansions vs 2,104 expansions). Unfortunately, some heuristics cannot be directly compared to one another using the dominance relationship (for example, a disjoint pattern database with the bottom 6 disks and the top 6 disks does not dominate, nor is it dominated by, a single pattern database with 8 disks). We make no predictions when neither heuristic dominates the other. When attempting to select a heuristic for the Towers of Hanoi with square costs, we make the correct decision 23/60 times, and when using reverse square costs, we make the correct decision 13/60 times.

As we can see, GDRC is a much more effective metric for selecting between heuristics for GBFS, picking the correct PDB more than 90% of the time compared to the dominance metric, which cannot make predictions for some pairings of heuristics, and makes the correct prediction less than half of the time.

Building a heuristic by hill climbing on GDRC

Our experiments from the previous section suggest that it is possible to use GDRC to compare heuristics against one another. This implies that, as an automated quantitative metric, GDRC allows us to automatically construct effective abstraction-based heuristics for greedy search in many domains. For example, for 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. 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, a small fraction of the half billion node state space, and sampled 10% of these nodes) to establish d^* values for nodes, and the h value can be looked up in the pattern database. It is also possible to obtain a better estimate of GDRC by randomly selecting nodes and calculating h and d^* for those nodes. We then select the PDB with the highest GDRC. This process iterates until either all PDBs have a worse GDRC than the current one, or until the PDB has reached the desired size. While we elected to use hill climbing due to its simplicity, it is possible to use other methods to explore the space of abstractions.

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

Figure 8: Average expansions required to solve TopSpin problems

When used to generate a 6 disk PDB for TopSpin, this process 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 search and A*, but when we change the costs such that moving an even disk costs 1 and moving an odd disk costs 10, 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 pattern database for greedy 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 a heuristic for greedy search on such a problem, we end up with a heuristic that keeps the abstracted and the refined disks connected to one another. As can be seen in Figure 8, the Contiguous pattern database produced by hill climbing on GDRC works much better for greedy search as compared to the Big Operators pattern database that tracks only the expensive disks. A*, on the other hand, performs much better when using the Big Operators pattern database, because the average values found in this pattern database are much higher. We can see the importance of creating a good pattern database when we consider the Random row from Figure 8, which contains the average number of expansions from 20 different randomly selected pattern databases.

We also evaluated GDRC-generated PDBs for the sliding tile puzzle. For the sliding tile puzzle, we refined one tile at a time, selecting the tile that increased GDRC the most. The resulting pattern database tracked the 1, 3, 4, 7, 8, and 11 tiles. The results of using this PDB are shown in Figure 6. 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 search 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 3 orders of magnitude faster than the worst 6 tile PDB for GBFS. It also handily outperformed the state-of-the-art instance specific PDB construction technique.

GDRC can also be used to select a heuristic from among many heuristics. For example, as Figure 7 (top) shows, on the Towers of Hanoi domain by selecting the PDB with the highest GDRC, we automatically select the best PDB out of 19 for GBFS.

Related Work

While there has been much work on constructing admissible heuristics for optimal search, there has been much less work on heuristics for GBFS.

Most analyses of heuristics assume their use for optimal search. However, Hoffmann (2005) analyzes the popular delete-relaxation heuristic (Hoffmann and Nebel 2001) for domain-independent planning, showing that it leads to polynomial-time behavior for enforced hill-climbing search in several benchmarks. Hoffmann (2011) automates this analysis. Although enforced hill climbing is a kind of greedy search, it 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, which means that the algorithm only cares about how the heuristic performs in a small local region of the space. Observation 1 is also related to a point discussed by Hoffmann (2005), who points out that in many domains, it is possible to find a plan leading to a goal where the heuristic never increases, meaning there are no local minima in the space at all.

Wilt and Ruml (2012) use a GDRC-like metric to predict when Weighted A* and GBFS will fail to provide speed-up. We use GDRC to directly compare heuristics against one another, and to build or find quality heuristics for GBFS.

Xu, Fern, and Yoon (2010) discuss how to learn search guidance specifically for planners, relying on features of the state description. Arfaee, Zilles, and Holte (2011) discuss constructing an inadmissible heuristic for near-optimal search. Thayer and Ruml (2011) show how to debias an admissible heuristic into an inadmissible one. While these works discuss creating a heuristic, none of them address metrics or comparing multiple heuristics as we do here.

Wilt and Ruml (2014) show that large local minima harm the performance of best-first search, and that the expected size of local minima is smaller for a unit-cost (speedy search) heuristic. They argue that it is local minimum size that is most important for predicting how well greedy best-first search will perform. At first glance, that would seem to be at odds with the results of this paper, but it is worth noting that it is generally the case that domains with large local minima have poor GDRC. This is because, by definition, all paths from a node n in a local minimum will contain nodes with $h(n') > h(n)$, so the heuristic value of n is off by at least $h(n') - h(n)$. This additional error increases the likelihood that the heuristic will have a poor GDRC.

An alternative approach to solving problems with greedy best-first search is to use a different algorithm, one that is, by construction, better suited to the available heuristics. This approach is taken by algorithms that use randomness (Imai and Kishimoto 2011; Nakhost and Müller 2013; Valenzano et al. 2014).

Conclusion

We have shown several examples in which the conventional guidelines for building heuristics for A* can actually harm the performance of GBFS. These helped us discover attributes of the heuristic that help or harm the performance of GBFS. We then introduced the GDRC metric and explained how it captures the preferences we observed for GBFS heuristics. We showed that GDRC is a useful metric to compare different heuristics for greedy search, and demonstrated how it can be used to automatically construct appropriate heuristics for GBFS. Given the importance of greedy best-first search in solving large problems quickly, we hope this investigation spurs further analysis of suboptimal search algorithms and the heuristic functions they rely on.

Acknowledgments

We thank anonymous reviewers for their helpful suggestions and we gratefully acknowledge support from NSF (award 1150068).

References

Arfaee, S. J.; Zilles, S.; and Holte, R. C. 2011. Learning heuristic functions for large state spaces. *Artificial Intelligence* 175(16-17):2075–2098.

Culberson, J. C., and Schaeffer, J. 1998. Pattern databases. *Computational Intelligence* 14(3):318–334.

Dechter, R., and Pearl, J. 1988. The optimality of A*. In Kanal, L., and Kumar, V., eds., *Search in Artificial Intelligence*. Springer-Verlag. 166–199.

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

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

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

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

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

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.

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

Imai, T., and Kishimoto, A. 2011. A novel technique for avoiding plateaus of greedy best-first search in satisficing planning. In *Proceedings of AAAI*.

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

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

Korf, R. E., and Taylor, L. A. 1996. Finding optimal solutions to the twenty-four puzzle. In *Proceedings of AAAI*, 1202–1207.

Korf, R. E. Analyzing the performance of pattern database heuristics. In *Proceedings of AAAI*.

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

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

Korf, R. E. 1997. Finding optimal solutions to Rubik's cube using pattern databases. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Conference on Innovative Applications of Artificial Intelligence*, AAAI'97/IAAI'97, 700–705.

Nakhost, H., and Müller, M. 2013. Towards a second generation random walk planner: an experimental exploration. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, 2336–2342. AAAI Press.

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.

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

Thayer, J. T., and Ruml, W. 2011. Learning inadmissible heuristics during search. In *Proceedings of the Twenty-First International Conference on Automated Planning and Scheduling*.

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

Valenzano, R. A.; Sturtevant, N. R.; Schaeffer, J.; and Xie, F. 2014. A comparison of knowledge-based GBFS enhancements and knowledge-free exploration. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling, ICAPS 2014*.

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

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

Xu, Y.; Fern, A.; and Yoon, S. W. 2010. Iterative learning of weighted rule sets for greedy search. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling, ICAPS 2010, Toronto, Ontario, Canada, May 12-16, 2010*, 201–208.