Heuristics for Bounded-Suboptimal Search

Lior Siag, Ariel Felner, Shahaf S. Shperberg

Department of Software and Information Systems Engineering, Ben-Gurion University of the Negev, Israel
siagl@post.bgu.ac.il, {felner, shperbsh}@bgu.ac.il

Abstract

In heuristic search, it is well-established that different types of heuristics are suited for optimal heuristic search (OHS) and unbounded suboptimal search (USS). In OHS, the heuristic should minimize the error in estimating the true cost of the shortest path, whereas in USS, it is more beneficial for the heuristic to exhibit a clear gradient toward the goal, regardless of the error. However, no study has specifically investigated which heuristic is most effective for bounded suboptimal search (BSS), and the current standard is to use heuristics designed for OHS. This paper introduces a novel method for creating heuristics tailored to BSS by linearly combining heuristics that were designed for OHS and USS. Through experimental evaluation, the proposed method is compared with those suited for OHS and USS. The results demonstrate that, within certain suboptimality bounds, our new heuristic approach outperforms OHS and USS heuristics for various BSS algorithms.

Code & Appendix —
https://github.com/SPL-BGU/BSS-Heuristics

1 Introduction

In *optimal heuristic search* (denoted OHS), the aim is to find a least-cost path from some given $start$ state to a desired $goal$ state. In OHS, there are generally two tasks: (i) finding a solution and (ii) proving its optimality (Cushing, Benton, and Kambhampati 2010; Thayer and Ruml 2011). Finding a solution involves identifying a path that leads from some given initial state to a goal state. On the other hand, proving a solution's optimality means demonstrating that no other solution exists with a lower cost.

One of the most renowned algorithms in optimal search is A* (Hart, Nilsson, and Raphael 1968). A* utilizes the heuristic to avoid checking nodes where the optimal solution cannot go through. The difference between the true cost and the heuristic, also called the *heuristic error* is what we want the heuristic to minimize. The lower the error, the more nodes A* can prune and thus expand fewer nodes.

Proving a solution's optimality can be difficult or even infeasible, as it requires checking a large number of nodes. In many real-world applications, such as robotics and planning, finding an optimal solution can be computationally expensive. Instead, a solution that is close to optimal but obtained faster may be more practical. *Bounded-suboptimal search* (denoted BSS) produces a solution that is at most worse than the optimal solution by some factor w, called the *suboptimality bound*. A solution bounded by w is called w*-admissible*. Notable examples of such an algorithms are *Weighted A** (denoted WA*) (Pohl 1970) and *Improved Optimistic Search* (denoted IOS) (Chen et al. 2019).

On the extreme end, *unbounded-suboptimal search* (denoted USS) only aims at finding a solution and thus offers no guarantees on solution quality. An early but still popular algorithm for this is Greedy Best-First Search (GBFS) (Doran and Michie 1966). This algorithm uses only the heuristic as its node evaluation function.

In BSS and USS, unlike OHS, algorithms do not need to address task (ii)—proving the optimality of the discovered solutions. Instead, in BSS, they only need to verify that the returned solution falls within the suboptimality bound, which is significantly easier, especially for large w. In USS, no quality verification is required at all. As a result, a heuristic with a small error does not necessarily reduce the search effort in BSS and USS. This means that choosing such a heuristic might not always be the best choice for BSS or USS. Wilt and Ruml (2016) introduced the *Goal Distance Rank Correlation* (GDRC), a measure of the correlation between heuristic values and the minimal number of steps needed to reach $goal$. They argued that GDRC is a more effective metric to optimize when designing heuristics for GBFS. Specifically, they demonstrated that heuristics with higher error but greater GDRC lead to fewer node expansions in GBFS compared to heuristics with lower error.

While prior work has focused on heuristics for OHS and USS, less attention has been given to BSS. In this work, we introduce a method for generating heuristics specifically suited for BSS, that balances solution proving and solution finding. In particular, we propose utilizing a linear combination of a heuristic that minimizes heuristic error and another one that maximizes GDRC. As w grows, the solution-proving task becomes easier, and when w is very large, it is trivial. Therefore, when combining both OHS and USS heuristics, we expect that as w grows, there would need to be more emphasis on the USS heuristic until we solely use

it for very large values of w.

Empirical evaluations across multiple domains, using both WA* and IOS, demonstrate that our new linear combination outperforms either heuristic alone for many suboptimality bounds. In addition, as a byproduct of the empirical evaluation, the accuracy of the GDRC metric is evaluated. While it is accurate in two tested domains, it fails to predict the performance of GBFS in the Sliding-Tile Puzzle domain, which utilizes its respective heuristic, suggesting that a more accurate metric may be needed.

2 Definitions, Notations, and Background

A heuristic search problem instance $(G = \langle V, E\rangle, c, start, goal, h)$ is composed of graph G with vertices V and edges E, c is a cost function $c : E \rightarrow \mathbb{R}^+$ that assigns costs to graph edges. The instance specifies a starting state ($start$) and a target state ($goal$) or a predicate $P : V \rightarrow 0, 1$ indicating whether a state satisfies goal conditions. h is a heuristic function that assigns to each state s an estimate of the cost associated with the least-cost path leading from s to the nearest goal state, often termed the "cost-to-go". The primary objective in heuristic search is to discover a path within graph G that connects $start$ to $goal$. The quality of the derived path is the cumulative cost of its constituent edges, determined by the cost function. We denote the cost of a path I as $C(I)$, the cost of the shortest path between $x, y \in V$ as $C(x, y)$, and $C(start, goal) = C^*$. A heuristic is called *admissible* if $\forall x \in V : h(x) \leq C(x, goal)$ and *consistent* if $\forall x, y \in V : h(x) \leq C(x, y) + h(y)$.

While the conventional aim is often to find the least-cost path, which is also referred to as the optimal path, there are scenarios where practical considerations lead to relaxing the optimality requirement. To this end, we consider a more general problem instance definition, $(G, c, start, goal, h, w)$, where w is a *suboptimality bound*, which is a multiplicative bound on the optimality of the returned solution. Formally, a solution to a problem instance is a path $I = (s_1, \ldots, s_n)$, where $s_1 = start$, $s_n = goal$ and $C(I) \leq w \cdot C^*$.

The distance d between two nodes is defined on G utilizing an alternative cost function $\forall (x, y) \in E : c'(x, y) = 1$, and $d(x, y) = C'(x, y)$. In other words, $d(x, y)$ is the minimal number of edges in any of the paths between x and y, denoted here as *hops-to-go*.[1]

2.1 A*, WA*, and GBFS

A* is an OHS algorithm given an admissible heuristic (Hart, Nilsson, and Raphael 1968). It is composed of two structural parts called OPEN and CLOSED. OPEN represents all nodes that are currently being evaluated in order to choose the next one to expand and generate its successors. CLOSED is used to detect repeating nodes, called *duplicates*. Duplicates can be (optionally) disregarded if the newly discovered path is *worse than or equal to* the known one. If it is better, we *reopen* the node by moving it from CLOSED back to OPEN. A *re-expansion* occurs when we expand a reopened node.

[1]We prefer to use this term and not the term *distance-to-go* (Ruml and Do 2007) because the latter may sound ambiguous.

$g(n)$ is the cost of the path from $start$ to n with minimum cost found so far by the search algorithm. A*'s node evaluation function (also called priority function) is defined as $f(n) = g(n) + h(n)$. When given a consistent heuristic, each node is expanded with $g(n) = C(start, n)$, and thus is never reopened (Pearl 1984).

A*, given a consistent heuristic, must expand all nodes n with $f(n) < C^*$ to prove the solution's optimality (Dechter and Pearl 1985). This means that a good heuristic is one that results in $f(n)$ as high as possible while still maintaining admissibility. In this case, the best heuristic is the true-cost heuristic (perfect heuristic). This gives us a metric by which we can judge the accuracy of a given heuristic. This metric can be formulated as the mean of $\forall x \in V : h(x)/C(x, goal)$ (or approximated using a subset of V). The closer the heuristic is to the true cost, the closer this metric is to 1. Note that while a heuristic closer to the true-cost heuristic (while remaining admissible) generally improves A*'s performance, there are pathological cases where it can lead to worse performance (Holte 2010). However, such cases are uncommon in practice.

*Weighted A** (WA*) (Pohl 1970) is created by putting greater emphasis on the heuristic in A*'s priority function. The nodes are ordered based on $f(n) = g(n) + w \cdot h(n)$. Remember that WA* only needs to prove that the solution cost is $\leq w \cdot C^*$, which means it needs to check significantly fewer nodes in order to prove its w-admissibility. This, in turn, allows it to give more weight to the "cost-to-go" instead of the cost incurred so far, thus moving more aggressively towards a solution. WA* is w-admissible given an admissible heuristic (Pearl 1984) and does not require reopenings given a consistent heuristic to still be w-admissible (Likhachev, Gordon, and Thrun 2003).

In USS, one can disregard $g(n)$ and only look to get closer to $goal$. GBFS is such algorithm where $f(n) = h(n)$ (tie-breaking in favor of lower g-values) (Doran and Michie 1966). This algorithm has no guarantees on the solution length in relation to C^*.

The three aforementioned algorithms lie on the same continuum, where we start with $w = 1$ and increase it up to $w = \infty$ (disregarding $g(n)$).

An equivalent formulation of WA* is that instead of inflating $h(n)$ by w, we now shift the focus to lowering $g(n)$. With this formulation, we get the following function:

$$f_\lambda(n) = \lambda \cdot g(n) + h(n) \quad (1)$$

It is easy to see that when $\lambda = 1$, we get the priority function of A*, and when $\lambda = 0$, we get the priority function of GBFS. This function also encompasses all WA* priority functions. For example, $\lambda = 0.5$ produces the equivalent of $g(n) + 2 \cdot h(n)$. In general, if we set $\lambda = 1/w$, nodes are prioritized exactly in the same manner and WA* is now $1/\lambda$-admissible.

2.2 Differentiating Solution Finding and Proving

The distinction between finding and proving a solution has been previously explored in the literature.

Algorithm 1: Improved Optimistic Search

Input: $start$, $goal$, w
Output: Solution path or failure

```
1  Push(start, OPEN)
2  Push(start, FOCAL)
3  I ← ∅                                   // C(I) = ∞
4  while C(I) is not w-optimal do
5      if I == ∅ then
6          Expand best from FOCAL
7          if best == goal then
8              I ← path(best)
9      else
10         Expand best from OPEN
11         foreach child s of best do
12             if s ∈ I ∧ found better path to s then
13                 Update cost of I
14 return failure
```

Focal Search A^*_ϵ (Pearl and Kim 1982) is a BSS algorithm which has two queues, OPEN which contains all nodes, and FOCAL which contains nodes n with $f(n) < w \cdot f_{min}$. By expanding nodes only from FOCAL, the algorithm is guaranteed to return a w-admissible solution. Thus OPEN is used to prove the solution's quality, and FOCAL is used to find one, as any node prioritization can be used, including a non-admissible one.

Optimistic Search Another BSS algorithm that emphasizes the distinction between these two tasks while also taking into consideration weaker aspects of WA* is *Optimistic Search* (Thayer and Ruml 2008). Optimistic Search initially finds a solution path I by running WA* with a $w_f = 2 \cdot w - 1$ where w is the suboptimality bound. The solution-finding search is not limited to admissible heuristics and can utilize non-admissible heuristics, though in their implementation, the same heuristic was used for both finding and proving the solution. After finding the path I, Optimistic Search interleaves node expansions between OPEN and FOCAL. It selects a node from Focal if the minimum priority in Focal is lower than the Focal priority of the node representing the current best solution (which is initially set to infinity); otherwise, it expands from OPEN. In some cases, Optimistic Search may find a solution whose cost exceeds the suboptimality bound. If the selection condition is not met, Optimistic Search continues expanding nodes from Open, potentially behaving like A* by expanding only from Open until the optimal solution is found.

f_{min} which is the lowest f-value in OPEN is a lower bound on C^*, and thus once $C(I) \leq w \cdot f_{min}$, the search terminates. In the worst-case scenario where the solution found is not w-admissible, Optimistic Search performs a full run of A* and returns an optimal solution.

Improved Optimistic Search IOS, as the name suggests, is an algorithm that stems from a set of improvements over Optimistic Search (Chen et al. 2019). IOS makes three crucial changes over Optimistic Search: (1) The priority function of the initial WA* search is $f(n) = g(n)/w_f + h(n)$. With this formulation, given a consistent heuristic, $f(n)$ of every node expanded becomes a lower bound on C^*. This allows for a more accurate initial estimation of C^* when attempting to prove the solution, and thus leads to earlier termination. (2) It does not re-expand nodes in either search, but keeps track of improvements found to the path found by the initial search. (3) The order of the finding and proving searches has been made entirely sequential, first finding a solution and then proving it. While it does increase the risk of finding a non-w-admissible solution, it empirically never happened due to keeping w_f suggested by Optimistic Search. The pseudocode of IOS, taken from the original paper, is shown in Algorithm 1.

Explicit Estimation Search EES (Thayer and Ruml 2011) is a search algorithm that maintains three queues: an OPEN queue sorted by $\hat{f}$, which may rely on a non-admissible heuristic; a FOCAL queue sorted by $\hat{d}$, estimating the number of hops (edges, independent of cost) to the goal; and a secondary OPEN queue, referred to as *cleanup*, ordered by f using an admissible heuristic to guarantee bounded suboptimality. These queues enable the algorithm to dynamically switch between solution discovery, refinement, or proof as the search progresses.

2.3 Heuristics in USS

Wilt and Ruml (2016) showed that heuristics effective for A* can be ill-suited for GBFS. This can lead to GBFS expanding more nodes than A* despite having no optimality guarantees. This is not the first time heuristics estimating the "cost-to-go" have been observed to perform poorly on GBFS. An alternative approach suggests using a heuristic that estimates "hops-to-go" instead of "cost-to-go" for GBFS, leading to the development of *Speedy Search* (Ruml and Do 2007).

The utilization of a heuristic that estimates "hops-to-go" has also been addressed in the context of BSS. A^*_ϵ was evaluated on the *Traveling Salesman Problem*, which had two heuristics: h_f, which is the number of remaining cities, and h_p, which is the number of remaining cities times the minimum cost to reach a city. Utilizing h_f for FOCAL showed improvement over using h_p (Pearl and Kim 1982). The usage of heuristics that evaluate "hops-to-go" was also evaluated in different algorithms such as Dynamic Weighted A* (Thayer and Ruml 2009), and Explicit Estimation Search (Thayer and Ruml 2011) to varying degrees of success.

When looking for a good heuristic for USS, we want a heuristic whose minimization will lead us towards $goal$ the fastest. Therefore, we do not care about its error, but instead we focus on the gradient towards $goal$, which by minimizing the heuristic, we will also get closer to $goal$. In order to judge how fitting a heuristic is to GBFS (and USS in general), a quantitative metric called *Goal Distance Rank Correlation* (GDRC) was created (Wilt and Ruml 2016). This metric measures the correlation between $h(x)$ and $d(x, goal)$. The core logic behind this metric is that when there is a correlation, nodes with small h will have a small

number of steps to $goal$ and thus lead to finding a solution faster. Similarly, when there is no correlation between h and d, low GDRC implies that h is not a good indicator for getting close to $goal$. Thus, expanding nodes with smaller h would not necessarily bring us closer to a solution, and GBFS would perform poorly with that heuristic.

The correlation between the two is calculated using Kendall's τ (Kendall 1938), though it was noted by Wilt and Ruml (2016) that τ-b (Kendall 1945) is more suited when handling ties, and thus will be used in this work.

We denote a heuristic that aims to maximize GDRC —and is therefore well-suited for GBFS—as h_f for solution *finding*, and a heuristic that aims to minimize the error (or equivalently, maximizes accuracy)—making it suitable for A* and optimal search in general—as h_p for quality *proving*.

3 Generating Heuristic for BSS

Through Equation 1 we can see that WA* lies between OHS and USS when $\lambda \in (0, 1)$. While either extreme prefers to utilize a single heuristic for its purposes, h_p in A* and h_f in GBFS, WA* is somewhere in between. This leads to the main idea of this paper: combining both h_p and h_f when assigning priority to a node in the context of WA* as follows. Let us define h_ϵ as a heuristic created by linearly combining h_p and h_f, given $\epsilon \in [0, 1]$, for some node x:

$$h_\epsilon(x) = \epsilon \cdot h_p(x) + (1 - \epsilon) \cdot h_f(x) \tag{2}$$

Since we would like to use h_ϵ in BSS algorithms, which often assume an admissible heuristic, we show that a linear combination of two admissible heuristics is admissible as well.

Lemma 1. *If h_p and h_f are both admissible, then h_ϵ is also admissible.*

Proof. Let $x \in V$, and $\epsilon \in [0, 1]$.

$$\begin{aligned} h_\epsilon(x) &= \epsilon \cdot h_p(x) + (1 - \epsilon) \cdot h_f(x) \\ &\leq (\epsilon + (1 - \epsilon)) \cdot \max\{h_p(x), h_f(x)\} \\ &= \max\{h_p(x), h_f(x)\} \leq C(x, goal) \end{aligned}$$

□

h_ϵ also preserves heuristic consistency, a property that enables algorithms like WA* and IOS to avoid node reopening.

Lemma 2. *If h_p and h_f are both consistent, then h_ϵ is also consistent.*

Proof. Let $x, y \in V$ be any two nodes, and $\epsilon \in [0, 1]$. Since h_f and h_p are consistent:

$$\begin{aligned} h_p(x) - h_p(y) &\leq C(x, y) \\ \epsilon \cdot (h_p(x) - h_p(y)) &\leq \epsilon \cdot C(x, y) \end{aligned}$$

and:

$$\begin{aligned} h_f(x) - h_f(y) &\leq C(x, y) \\ (1 - \epsilon) \cdot (h_f(x) - h_f(y)) &\leq (1 - \epsilon) \cdot C(x, y) \end{aligned}$$

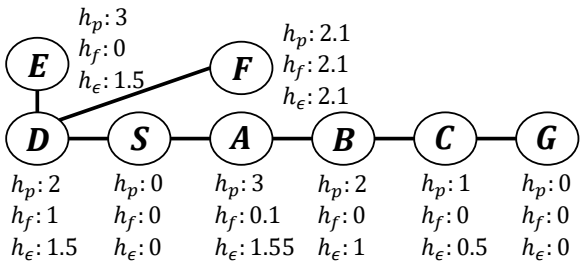


Figure 1: Example graph with h_p and h_f where $\epsilon = 0.5$

Alg	h	S	A	B	C	D	E	F	G
GBFS	h_f	0	0.1	0	0	1	0	2.1	0
	h_p	0	3	2	1	**2**	3*	2.1	0
	h_ϵ	0	1.55	1	0.5	**1.5**	**1.5**	2.1	0
A*	h_f	0	1.1	2	3	**2**	**2**	4.1	4
	h_p	0	4	4	4	**3**	5	4.1	4
	h_ϵ	0	2.55	3	3.5	**2.5**	**3.5**	4.1	4
WA*	h_f	0	1.2	2	3	**3**	**2**	6.2	4
	h_p	0	7	6	5	**6**	8	**6.2**	4
	h_ϵ	0	4.1	4	4	**4**	5	6.2	4

*Might be expanded before A, depending on tie-breaking

Table 1: Node priorities $f(x)$ for the example in Figure 1

Adding both inequalities and reordering results in:

$$\begin{aligned} &(\epsilon \cdot h_p(x) + (1 - \epsilon) \cdot h_f(x)) - \\ &\quad (\epsilon \cdot h_p(y) + (1 - \epsilon) \cdot h_f(y)) \\ &\qquad \leq (\epsilon + (1 - \epsilon)) \cdot C(x, y) \end{aligned}$$

Therefore:

$$h_\epsilon(x) - h_\epsilon(y) \leq C(x, y)$$

□

Since h_ϵ is admissible and consistent, it can be efficiently utilized by BSS algorithms, including WA* and IOS.

3.1 Example of h_ϵ Utilization

To illustrate the effectiveness of h_ϵ, we present a comparative example against h_f and h_p. We look at three algorithms: GBFS with $f(x) = h(x)$, A* with $f(x) = g(x) + h(x)$, and WA* with $w = 2$, thus $f(x) = g(x) + 2 \cdot h(x)$. In this example, we set $\epsilon = 0.5$. Each of these algorithms is executed 3 times, one for each of the heuristics for a total of 9 runs.

The example in Figure 1, illustrates a search problem where the aim is to find a path from S to G. All edges have a weight of 1. Each node has three heuristic values: h_p which is the proving heuristic (good for OHS), h_f which is the finding heuristic (good for GBFS) h_ϵ which is the combination (good for BSS). In this example, all algorithms have to expand nodes A, B, C, and G. Therefore, the difference in the algorithm performance is measured in how many nodes from

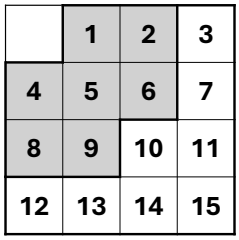


Figure 2: Ridge PDB for 15-STP, Ridge0 is colored in gray

D, E, F were expanded. Table 1 summarizes the node priorities $f(x)$ for all nodes under each algorithm and heuristic. The nodes from D, E, F that are expanded are highlighted in bold.

GBFS. When GBFS uses h_f the search immediately goes towards the solution and never expands D, E, or F. By contrast, when GBFS uses either h_p or h_ϵ it leads to node D being expanded and thus does more expansions than utilizing h_f.

A*. With h_p, A* expands D immediately after S, then only nodes on the path to $goal$, as their priority (4) is lower than that of E or F. With h_f, A* expands A, then B, D, and E in some order before C, causing one extra expansion. Similar behavior occurs with h_ϵ, though E now shares the same priority as C instead of B.

WA*. When WA* uses h_ϵ it expands D, but E and F are never expanded. WA* with h_p expands both D and F before expanding nodes on the path from A to G. WA* with h_f results in the expansion of both D and E before expanding G. This means h_ϵ manages to expand one less node than both h_f and h_p.

This example demonstrates that different heuristics impact search efficiency across various algorithms. For GBFS, h_f proves to be best as it directs immediately to $goal$. In contrast, A* performs best with h_p as it maximizes the nodes that do not need to be expanded. WA* demonstrates that utilizing h_ϵ in BSS can indeed perform better than either component alone and finds a solution with the fewest number of expansions.

4 Empirical Evaluation

The aim of this empirical evaluation is to assess the potential improvement of utilizing h_ϵ instead of h_p (or h_f) in BSS and study the effect of different values for ϵ as a function of w. It is important to note that for all our domains, the heuristics we used (see below) were such that h_p dominates h_f as it contains the information expressed by h_f and adds upon it.

In general, we expect to see that as ϵ becomes smaller (i.e., h_ϵ transitions from h_p to h_f), the heuristic accuracy decreases while the GDRC increases. Likewise, the smaller ϵ is, the closer it is to h_f and thus is more suited to GBFS and to large values of w. Wilt and Ruml (2016) suggest that such heuristic will have a higher GDRC, so we expect to see GBFS performs best with small ϵ. On the other hand, when ϵ is large, the heuristic is closer to h_p and will be more suited for small values of w.

4.1 Experimental Settings

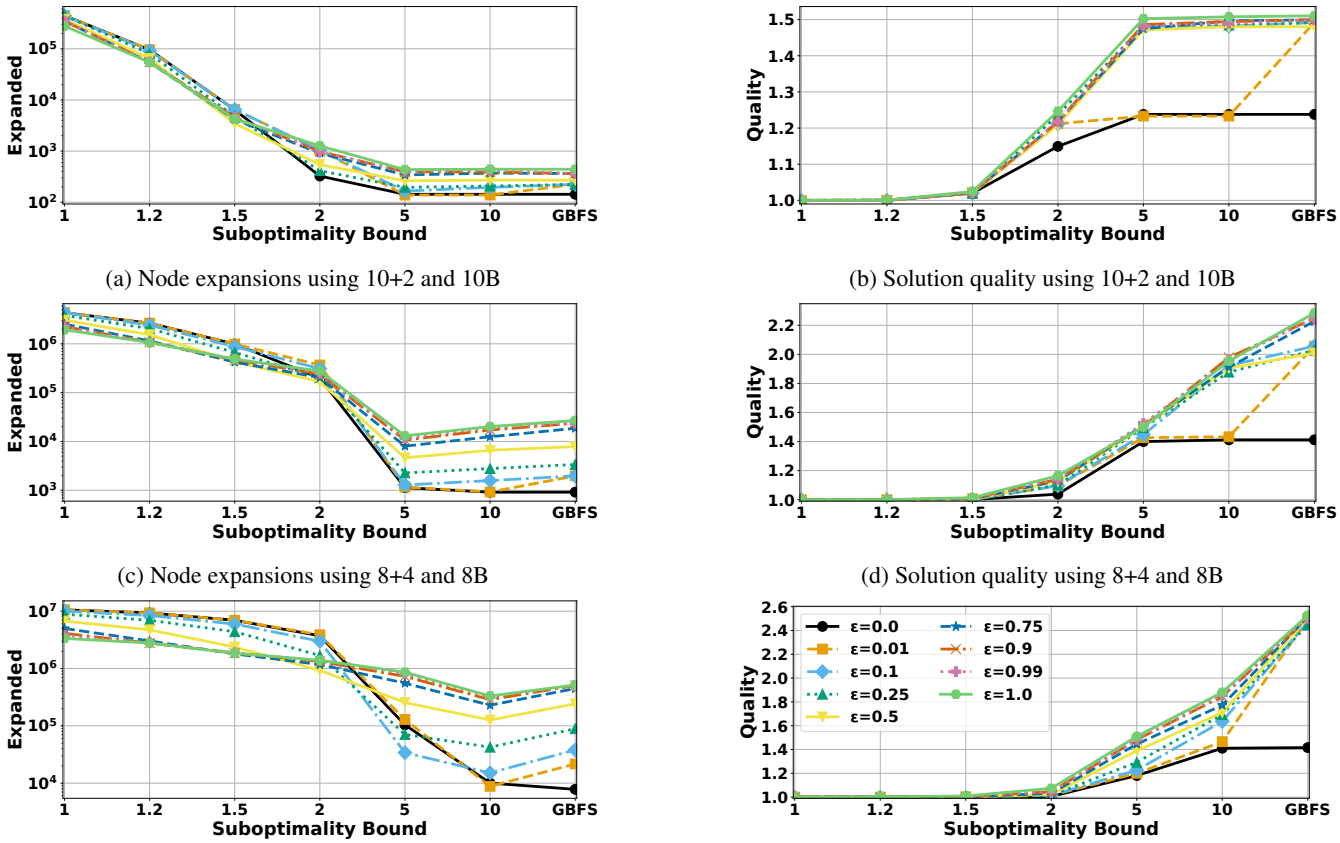
We experimented on the following domains:

4-Peg Towers of Hanoi (ToH). We use 12 disks, with an additive pattern database (PDB) (Felner, Korf, and Hanan 2004) of 10 bottom disks and 2 top disks, denoted 10+2, as well as 8+4, and 6+6 for h_p. For h_f, we use a PDB of only the bottom disks as suggested by Wilt and Ruml (2016), denoted 10B, 8B, and 6B. Indeed, the additive PDB has larger values and is more suited for h_p. However, the single bottom PDB has a much better gradient towards $goal$ than the additive PDB. The reason is that when the large disks move in ToH it is a better indication that we get closer to the goal than when the small disks move. Thus, while the single bottom PDB has smaller values than the additive PDB, it is more suitable for h_f than the additive PDB. 100 random $start$ instances were tested with the canonical $goal$ (where all disks are on a single right-most peg).

15 Sliding Tile Puzzle (STP). We use the 100 Korf's problems (Korf 1985) where the blank is in the top left corner in the solution. The heuristic used is an additive PDB illustrated in Figure 2, which is composed of two parts: Ridge0 and Ridge1. Ridge1 (far from blank) corresponds to the tiles $\{3, 7, 10, 11, 12, 13, 14, 15\}$ while Ridge0 (close to blank) corresponds to the rest, i.e., $\{1, 2, 4, 5, 6, 8, 9\}$. Both PDBs contain the blank (and do not compress it away). The additive *Ridge* heuristic is the addition of Ridge0 and Ridge1. In this domain, h_p is Ridge (adding both Ridge0 and Ridge1) while h_f uses Ridge1 only. In the Ridge heuristic, we separate tiles based on either close-to-blank or far-from-blank. Generally, the far tiles have to be sorted before the closer ones can be sorted. This means that Ridge1 will direct the search relatively quickly towards $goal$, which will only leave the tiles close to the blank to be sorted. Thus, using Ridge1 alone is more suited for h_f. Naturally, adding both Ridge0 and Ridge1 is more suited for h_p. This has the same effect as the top disks versus the bottom disks in ToH above.

Weighted 15 Sliding Tile Puzzle (WSTP). Here, the cost of moving a tile equals the number on it, e.g., the cost of moving tile 8 is 8. The heuristics are: *Manhattan Distance* (MD), which measures the number of moves needed, for h_f, and: *Weighted Manhattan Distance* (WMD), which counts the number of moves for each tile but multiples it by the tile's number, for h_p. Here, MD is an estimate on the minimal "hops-to-go" as it treats all edges as unit-cost, and thus is a good candidate for h_f. On the other hand, WMD adds information on top of MD, which may distort the gradient, but has significantly higher accuracy, thus it is more suitable for h_p. Again, we use 100 Korf's problems for this domain.[2]

[2]The costs of the optimal solutions were obtained using more resources than the limit given in this paper.



(a) Node expansions using 10+2 and 10B

(b) Solution quality using 10+2 and 10B

(c) Node expansions using 8+4 and 8B

(d) Solution quality using 8+4 and 8B

(e) Node expansions using 6+6 and 6B

(f) Solution quality using 6+6 and 6B

Figure 3: Results for A*, WA*, and GBFS in ToH — averaged over all instances

4.2 Algorithms, Weights, and values of ϵ

A*, WA*, and GBFS are all tested as parts of the same continuum. WA* is tested with varying weights of: 1.2, 1.5, 2, 5, 10. Weights of 20 and 50 are also used for STP and WSTP to show a clearer gradient towards GBFS, especially with small ϵ. IOS is also tested while utilizing h_ϵ as the heuristic for the initial solution finding search and h_p as the proving heuristic. This allows us to utilize the new method for faster solution finding while also having a better-suited heuristic for proving the solution's w-admissibility. The values for ϵ that are tested are 1 (which is pure h_p), 0.99, 0.9, 0.75, 0.5, 0.25, 0.1, 0.01, and 0 (which is pure h_f). All algorithms are tie-broken in favor of higher g-values.

When utilizing e.g., 10+2 as h_p and 10+0 as h_f in WA*, the bottom 10 disks will always be multiplied by w as $\epsilon \cdot w + (1-\epsilon) \cdot w = w$ (they are part of both h_p and h_f). By contrast, the top 2 disks will be only multiplied by $w \cdot \epsilon$ (they are only part of h_p).

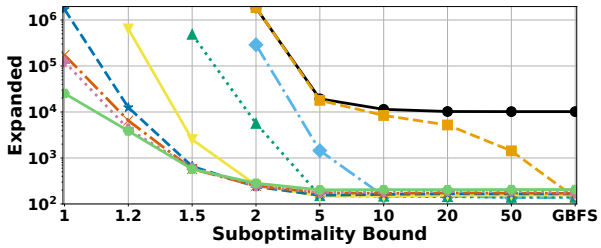
4.3 Metrics

The algorithms' performance is measured based on two metrics: *nodes expanded* (denoted Exp.), which is the number of nodes expanded by the algorithm until termination, and *solution quality* (denoted Qua.), which is achieved by dividing the resulting solution by C^*. The heuristics were evaluated based on their heuristic accuracy, denoted by h/C^*, and their GDRC. The higher the accuracy of the heuristic, the better performance we expect on A*, and the higher the GDRC, the better performance we expect on GBFS. We also expect that as w increases, the ϵ which would minimize expansions would be smaller.

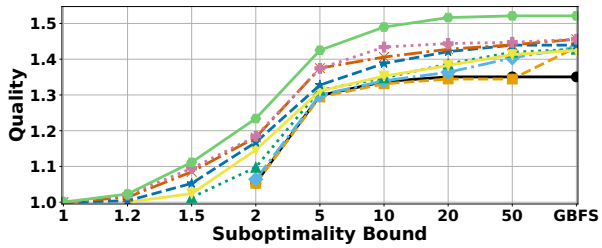
ϵ	STP		WSTP		ToH					
	Ridge + Ridge1		WMD + MD		10+2 + 10B		8+4 + 8B		6+6 + 6B	
	h/C^*	GDRC	h/C^*	GDRC	h/C^*	GDRC	h/C^*	GDRC	h/C^*	GDRC
1.00	**0.850**	**0.736**	**0.745**	0.530	**0.669**	0.799	**0.482**	0.610	**0.421**	0.395
0.99	0.847	0.720	0.739	0.529	0.668	0.799	0.481	0.618	0.419	0.416
0.90	0.812	0.721	0.680	0.529	0.665	0.799	0.472	0.618	0.399	0.416
0.75	0.754	0.723	0.582	0.531	0.660	0.799	0.458	0.631	0.367	0.437
0.50	0.658	0.712	0.419	0.540	0.651	0.809	0.434	0.674	0.314	0.494
0.25	0.561	0.663	0.255	0.543	0.642	0.810	0.410	0.696	0.260	0.534
0.10	0.503	0.622	0.157	0.568	0.636	0.810	0.395	0.699	0.228	0.565
0.01	0.469	0.620	0.098	0.578	0.633	0.810	0.387	0.699	0.209	0.573
0.00	0.465	0.602	0.092	**0.590**	0.632	**0.827**	0.386	**0.731**	0.207	**0.621**

Table 2: h/C^* and GDRC for every heuristic combination

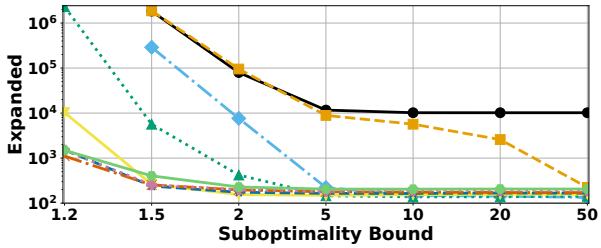
Table 2 presents the heuristic accuracy and GDRC across all domains, heuristics, and ϵ values. In general, decreasing ϵ results in lower heuristic accuracy but improved GDRC, reflecting the shift towards h_f. An exception to this is Ridge in STP, where using $\epsilon = 1$ (i.e., relying solely on h_p) yields



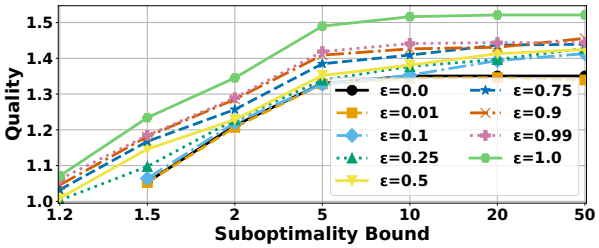
(a) Nodes expanded for A*, WA*, and GBFS



(b) Solution quality for A*, WA*, and GBFS



(c) Nodes expanded for IOS



(d) Solution quality for IOS

Figure 4: Results for STP — averaged over all instances

both the highest accuracy and the best GDRC. This indicates using $\epsilon = 1$ will lead to better results not only for A*, but also for GBFS.

Each instance run—defined by domain, *start*, *goal*, heuristic, and algorithm configuration—is given a 64 GB memory limit that upon reaching, the instance is considered a failure. If an algorithm fails to solve at least one problem instance in a given domain, we omit that data point from the figure. The full numerical tables for all domains, including additional graphs, appear in the appendix.

4.4 4-Peg Towers of Hanoi (ToH)

Figure 3 presents the results for A*, WA*, and GBFS for ToH. For WA*, across all three additive PDB configurations, it is most useful to use h_p when $w = 1$ (i.e., A*) and h_f for GBFS. Starting from 1.5 (1.2 for 6+6), we can see a shift towards utilizing ϵ between 0.25-0.75, meaning that a combination of h_p and h_f provides a better heuristic. This means that for those weights, there is still a need to prove the solution's quality, but not as strong as with $w = 1$, making solution finding a needed task as well. With 10+2, the trend of decreasing ϵ (when w becomes larger) leads to better results and ends in $w = 2$ where $\epsilon = 0$ is best, but the trend continues longer for 8+4, which ends in $w = 5$, and 6+6, which continues up to $w = 10$. IOS shows similar trends across all three PDB configurations, but due to limited space, the figure was omitted from the paper and appears in the appendix.

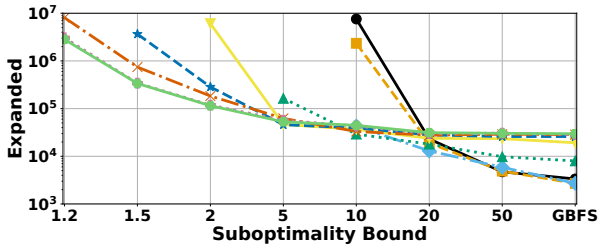
Tie-breaking and node ordering can significantly impact the number of expanded nodes. Observe GBFS with heuristics 10+2 and 10B and epsilons 0.01 and 0. The largest value that is provided by the small PDB is 3 and thus its effect on the priority function can be considered a tie-breaker between nodes with equal large PDB value. We expect their performance to be the same, since there is not a lot of information lost between the two. However, the small PDB can change the order in which nodes are expanded, which can then lead to different successor generations. Thus, the two GBFS searches pursue different paths, which then leads to different solutions found, as well as different performance.

Interestingly, the least-informed heuristics ($\epsilon \in \{0, 0.01\}$) often yield the best solution quality. This effect is particularly pronounced for $w = \{5, 10\}$, where they significantly outperform $\epsilon \geq 0.1$. This suggests that prioritizing the placement of the bottom disks early leads to better-quality solutions. Once these disks are in place, the search can focus on the remaining disks, which have little to no effect on the priority, resulting in a near-uniform-cost search for the remaining disks, leading to a good solution quality.

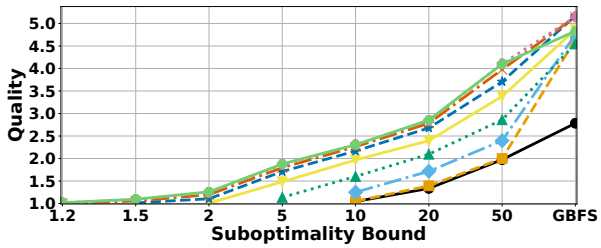
4.5 15 Sliding Tile Puzzle (STP)

STP results are provided in Figure 4. Here, again, we see a trend similar to that in ToH, where as w increases, smaller values of ϵ are preferable. Note that towards the larger weights, and GBFS, the preferable value for ϵ is not $\epsilon = 0$ but rather $\epsilon = 0.01$. In this case, Ridge0, in the form of a tie-breaker, allows for prioritizing nodes better once the tiles represented by Ridge1 are ordered and give a low (or 0) heuristic value. IOS again shows a similar trend to that of WA*. WA* starts showing a preference towards smaller ϵ when $w = 2$, while IOS shows this preference much earlier, at $w = 1.2$ all the way up to $w = 50$. Here again, like in ToH, we see that prioritizing moving quickly towards the solution, and then performing a near-uninformed search in that area of the search leads to good solution quality.

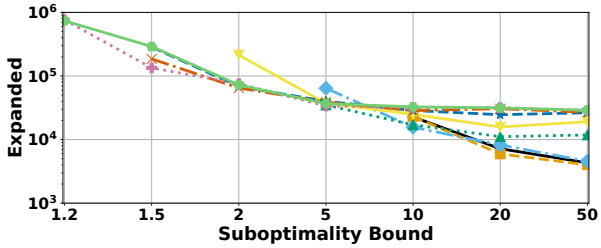
Note that as GDRC decreases (i.e., a lower correlation between h and d), the performance of GBFS improves (excluding $\epsilon = 0$). This suggests that higher GDRC does not necessarily lead to better GBFS performance. This is a problem that stems from solution-finding and not proving. We can see that when w is low, e.g., 1.2 or 1.5, the behavior of



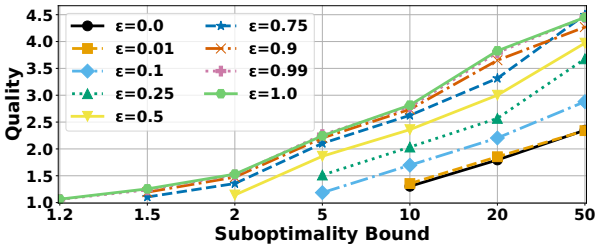
(a) Nodes expanded for A*, WA*, and GBFS



(b) Solution quality for A*, WA*, and GBFS



(c) Nodes expanded for IOS



(d) Solution quality for IOS

Figure 5: Results for WSTP — averaged over all instances

$\epsilon \in \{0, 0.01\}$ for both WA* and IOS is comparable. But, as the proving of solution optimality becomes trivial, the gap in performance widens significantly. Furthermore, while there is a relatively small difference between the GDRC of epsilons 0.01 and 0.00 (0.620 and 0.602, respectively), the difference in the performance of GBFS between these two values of ϵ is two orders of magnitude. This means that the differences in GDRC also cannot predict the magnitude of the difference in performance between different heuristics. Ultimately, we suggest that GDRC should be used as a coarse metric to judge whether a heuristic might fit GBFS, but should not be used as a fine metric to differentiate between two heuristics with close GDRC values.

4.6 Weighted 15 Sliding Tile Puzzle (WSTP)

WSTP results are provided in Figure 5. Here, the trend noted previously also appears. Initially, when $w \leq 1.5$, proving the solution is most needed, but as the weight increases, we can see that decreasing ϵ leads to improved performance. This continues up to $w = 50$ for WA* and $w = 20$ for IOS where the best performing heuristic has $\epsilon \in \{0, 0.01\}$, i.e., h_f (or very close to it). As the task of proving optimality (or bounded-suboptimality) becomes negligible, the main task of the algorithm becomes finding a solution. Thus, being closer to MD is better as it is a better estimate for "hops-to-go" or how far we are from *goal* in terms of states over WMD.

5 Conclusions and Future Work

This work presents the idea of (linearly) combining heuristics fitting OHS (h_p) and heuristics fitting USS (h_f), to create improved heuristics for BSS (h_ϵ). The empirical evaluation shows that for each domain, there is a bounded-suboptimality range where h_ϵ is better than either h_p or h_f alone. This, of course, depends on the algorithm chosen and the heuristics used, but in general, as w increases, ϵ should decrease. Two prominent algorithms, WA* and IOS show similar trends. Using h_ϵ reduces the number of expansions by up to a factor of 3 over using h_f or h_p alone. The empirical evaluation also looked at the GDRC metric and concluded that it can be used as a coarse metric to judge whether or not a heuristic will be a good fit for USS. However, it should not be used as a fine metric that differentiates the better heuristic between two heuristics with close GDRC values.

We propose two key directions for future work. First, developing a more accurate metric to evaluate heuristic performance in the context of USS. Second, designing a method to generate heuristics optimized for BSS directly, rather than relying on a linear combination of existing heuristics.

Acknowledgments

This work was supported by the Israel Science Foundation (ISF) grant #909/23 awarded to Shahaf Shperberg and Ariel Felner, by Israel's Ministry of Innovation, Science and Technology (MOST) grant #1001706842, in collaboration with Israel National Road Safety Authority and Netivei Israel, awarded to Shahaf Shperberg, by BSF grant #2024614 awarded to Shahaf Shperberg, and by BSF grant #2021643 to Ariel Felner.

References

Chen, J.; Sturtevant, N.; Doyle, W.; and Ruml, W. 2019. Revisiting suboptimal search. In *Proceedings of the International Symposium on Combinatorial Search*, volume 10, 18–25.

Cushing, W.; Benton, J.; and Kambhampati, S. 2010. Cost

Based Search Considered Harmful. In *SOCS*, 140–141. AAAI Press.

Dechter, R.; and Pearl, J. 1985. Generalized Best-First Search Strategies and the Optimality of A*. *J. ACM*, 32(3): 505–536.

Doran, J.; and Michie, D. 1966. Experiments with the Graph Traverser program. *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences*, 294: 235 – 259.

Felner, A.; Korf, R. E.; and Hanan, S. 2004. Additive Pattern Database Heuristics. *J. Artif. Intell. Res.*, 22: 279–318.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Trans. Syst. Sci. Cybern.*, 4(2): 100–107.

Holte, R. C. 2010. Common Misconceptions Concerning Heuristic Search. In *SOCS*, 46–51. AAAI Press.

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

Kendall, M. G. 1945. The treatment of ties in ranking problems. *Biometrika*, 33(3): 239–251.

Korf, R. E. 1985. Depth-First Iterative-Deepening: An Optimal Admissible Tree Search. *Artif. Intell.*, 27(1): 97–109.

Likhachev, M.; Gordon, G. J.; and Thrun, S. 2003. ARA*: Anytime A* with Provable Bounds on Sub-Optimality. In *NIPS*, 767–774. MIT Press.

Pearl, J. 1984. *Heuristics - intelligent search strategies for computer problem solving*. Addison-Wesley series in artificial intelligence. Addison-Wesley.

Pearl, J.; and Kim, J. H. 1982. Studies in Semi-Admissible Heuristics. *IEEE Trans. Pattern Anal. Mach. Intell.*, 4(4): 392–399.

Pohl, I. 1970. Heuristic Search Viewed as Path Finding in a Graph. *Artif. Intell.*, 1(3): 193–204.

Ruml, W.; and Do, M. B. 2007. Best-First Utility-Guided Search. In *IJCAI*, 2378–2384.

Thayer, J. T.; and Ruml, W. 2008. Faster than Weighted A*: An Optimistic Approach to Bounded Suboptimal Search. In *ICAPS*, 355–362.

Thayer, J. T.; and Ruml, W. 2009. Using Distance Estimates in Heuristic Search. In *ICAPS*. AAAI.

Thayer, J. T.; and Ruml, W. 2011. Bounded suboptimal search: A direct approach using inadmissible estimates. In *IJCAI*, volume 2011, 674–679.

Wilt, C. M.; and Ruml, W. 2016. Effective Heuristics for Suboptimal Best-First Search. *J. Artif. Intell. Res.*, 57: 273–306.