Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling

Using Alternative Suboptimality Bounds in Heuristic Search

Richard Valenzano, Shahab Jabbari Arfaee
University of Alberta
{valenzan,jabbaria}@cs.ualberta.ca

Jordan Thayer
Smart Information Flow Technologies
jordan.thayer@gmail.com

Roni Stern
Harvard University
roni.stern@gmail.com

Nathan R. Sturtevant
University of Denver
sturtevant@cs.du.edu

Abstract

Most bounded suboptimal algorithms in the search literature have been developed so as to be ϵ-admissible. This means that the solutions found by these algorithms are guaranteed to be no more than a factor of $(1 + \epsilon)$ greater than optimal. However, this is not the only possible form of suboptimality bounding. For example, another possible suboptimality guarantee is that of *additive bounding*, which requires that the cost of the solution found is no more than the cost of the optimal solution plus a constant γ.

In this work, we consider the problem of developing algorithms so as to satisfy a given, and arbitrary, suboptimality requirement. To do so, we develop a theoretical framework which can be used to construct algorithms for a large class of possible suboptimality paradigms. We then use the framework to develop additively bounded algorithms, and show that in practice these new algorithms effectively trade-off additive solution suboptimality for runtime.

1 Introduction

When problem-solving in domains with large state spaces, it is often not feasible to find *optimal* solutions given practical runtime and memory constraints. In such situations, we are forced to allow for *suboptimal* solutions in exchange for a less resource-intensive search. Algorithms which may find such solutions are called *suboptimal algorithms*.

Some suboptimal algorithms satisfy a *suboptimality bound*, which is a requirement on the cost of any solution that is set *a priori* of any problem-solving. By selecting a suboptimality bound, a user defines the set of solutions which are considered acceptable. For example, where C^* is the optimal solution cost for a given planning task, the ϵ-admissible bound requires that any solution returned must be from the set of solutions whose cost C satisfies the relation $C \leq (1 + \epsilon) \cdot C^*$.

Notice that requiring an algorithm to be ϵ-admissible for some particular ϵ involves setting a maximum of $(1 + \epsilon)$ on the solution suboptimality, where suboptimality is measured by C/C^*. Since the development of the first ϵ-admissible algorithm, Weighted A* (WA*) (Pohl 1970), this suboptimality measure has been by far the most commonly used measure. For example, ever since 2008, the International Planning Competition (IPC) satisficing track scoring function has been based on the idea that the relative suboptimality of two plans is given by their ratio. The popularity of this suboptimality measure can also be seen in the fact that the majority of algorithms with suboptimality guarantees that have been developed since WA* have been ϵ-admissible, including A$^*_\epsilon$ (Pearl and Kim 1982), Optimistic Search (Thayer and Ruml 2008), and EES (Thayer and Ruml 2011). However, there also exists other ways to measure suboptimality and other types of suboptimality bounds. For example, one alternative to C/C^* is to measure suboptimality by $C - C^*$. A bound corresponding to this measure would then require that any solution returned must have a cost C such that $C \leq C^* + \gamma$ for some $\gamma \geq 0$.

In this paper, we consider such alternative suboptimality bounds so as to evaluate the generality of existing methods and to give more choice into how suboptimality guarantees can be specified. If a system designer wishes to have guarantees (or to use suboptimality measures) that are not based on ϵ-admissibility, they currently have no means to do so, and yet despite the popularity of this paradigm, it is not necessarily suitable in all cases. For example, consider a problem in which plan cost refers to the amount of money needed to execute the plan. The value (or *utility*) of money is known to be a non-linear function (*ie.* \$10 is more valuable to someone with no money than it is to a billionaire). In particular, assume that that for some individual the utility of a plan with cost C is given by $K - \log_2(2 + C)$ for some constant $K > 0$ and the desired guarantee on suboptimality is that the utility of any plan found will be no more than 10% worse than is optimally possible. While any solution found by a WA* instance parameterized so as to be 0.1-admissible is guaranteed to be at most 10% more costly than the optimal solution, clearly this does not actually correspond with the desired requirement on utility. Moreover, it is unclear how to parameterize WA* correctly so as to satisfy the given requirement without prior knowledge of C^*.

These issues raise the following question: how can we construct algorithms that are guaranteed to satisfy a given suboptimality bound? In this paper, we show that four different classes of existing algorithms can be modified in this end. These classes are anytime algorithms, best-first search algorithms, iterative deepening algorithms, and focal list based algorithms. As each of these algorithm paradigms is

Notation	Meaning
n_i	the initial node
$c(n, n')$	cost of the edge between n and n'
C^*	the cost of the optimal path
$pred(n)$	stored predecessor (or parent) of n
$g(n)$	cost of the best path from n_i to n found so far
$g^*(n)$	cost of optimal path between n_i and n
$h^*(n)$	cost of optimal path from n to a goal
$h(n)$	heuristic estimate of cost to nearest goal node h is *admissible* if $h(n) \leq h^*(n)$ for all n

Table 1: Heuristic search notation.

best-suited for different types of problems, by extending them all so as to be able to handle arbitrary bounding constraints, we are allowing a system designer to not only specify a desired form of bounding, but to also select the best search framework for their particular domain.

The contributions of this paper are as follows. First, we introduce a functional notion of a suboptimality bound so as to allow for the definition of alternative bounding paradigms. We then develop a theoretical framework which identifies how existing algorithms can be modified so as to satisfy alternative bounds. Finally, we demonstrate that the framework leads to practical algorithms that can effectively trade-off guaranteed solution quality for improved runtime when considered for additive bounds.

2 Algorithms for Arbitrary Bounds

In this section, we will generalize the notion of a *suboptimality bound* and consider ways in which we can modify existing algorithms so that they are guaranteed to satisfy large classes of possible bounds. Table 1 lists some notation that will be used throughout this paper.

2.1 Bounding Functions

Recall that the ϵ-admissible bound requires that the cost C of any solution returned must satisfy $C \leq (1 + \epsilon) \cdot C^*$ for a given $\epsilon \geq 0$. We generalize this idea by allowing for an acceptable level of suboptimality to be defined using a function, $B : \mathbb{R} \rightarrow \mathbb{R}$. This *bounding function* is used to define the set of acceptable solutions as those with cost C for which $C \leq B(C^*)$. This yields the following definition:

Definition 1 *For a given bounding function B, an algorithm A will be said to* satisfy *B if on any problem, any solution returned by A will have a cost C for which $C \leq B(C^*)$.*

As an example of how this definition applies, notice that the ϵ-admissible requirement corresponds to the bounding function $B_\epsilon(x) = (1 + \epsilon) \cdot x$, since an algorithm is ϵ-admissible if and only if it satisfies B_ϵ. Similarly, an algorithm is optimal if and only if it satisfies the bounding function $B_{opt}(x) = x$. Other bounding functions of interest include $B_\gamma(x) = x + \gamma$ for some $\gamma \geq 0$, which corresponds to an additive bound, and $B(x) = x + \log x$, which allows for the amount of suboptimality to grow logarithmically.

Notice that for all these bounding functions, $B(x) \geq x$ for all x. This is a necessary condition, as to do otherwise is to allow for bounding functions that require better than optimal solution quality. Any bounding function B satisfying this requirement will also be trivially satisfied by any optimal algorithm. However, selecting an optimal algorithm for a given B (where $B \neq B_{opt}$) defeats the purpose of defining an acceptable level of suboptimality, which was to avoid the resource-intensive search typically required for finding optimal solutions. The goal is therefore not only to find an algorithm that satisfies B, but to find an algorithm which satisfies B and can be expected to be faster than algorithms satisfying tighter bounds. In the remainder of this section, we do so in several well-known heuristic search frameworks.

2.2 Bounding with Anytime Algorithms

In this section, we will demonstrate that we can use anytime algorithms, regardless of the suboptimality paradigm they were initially developed for, to satisfy any monotonically non-decreasing bounding function. We then argue that this approach is problematic and that we instead need algorithms tailored specifically for the given bounding function.

So as to use an anytime algorithm to satisfy a monotonically non-decreasing bounding function B, we will require that during execution of the algorithm, there is a lower bound L on the optimal cost that is available. Let C be the cost of the incumbent solution. Since B is monotonically non-decreasing, this means that if $C \leq B(L)$ then $C \leq B(C^*)$. Therefore, we can use an anytime algorithm to satisfy B by only terminating and returning the incumbent solution when its cost satisfies $C \leq B(L)$.

For example, consider Anytime Weighted A* (AWA*) (Hansen and Zhou 2007). This algorithm runs WA* to find a first solution, and then continues the search so as to find better solutions. Hansen and Zhou showed that the lowest f-cost of all nodes on the open list offers a lower-bound on C^*, where $f(n) = g(n) + h(n)$ and h is admissible. After each node expansion, we can then use the value of this lower bound, L, and the cost of the incumbent solution, C, to check if $C \leq B(L)$ is true, in which case we can terminate. This technique was first considered by Thayer and Ruml (2008) who ran AWA* as an ϵ-admissible algorithm. To do so, they parameterize AWA* so that it can be expected to find an initial solution quickly, even if the cost of that solution does not satisfy $C \leq B_\epsilon(C^*)$. They then continue the search until the incumbent solution does satisfy this condition. Above, we have extended this technique so as to satisfy any monotonically non-decreasing B, even when the initial algorithm was designed for some other bounding function B'. This means that we can use AWA*, or any other ϵ-admissible anytime algorithm that can maintain a lower-bound on C^*, to handle other bounding functions.

However, the performance of AWA* is greatly affected by a parameter, called the *weight*, which determines the algorithm's greediness. While it is not too difficult to set the weight when AWA* is to be used as an ϵ-admissible algorithm, it is not clear how to set it — in general and without prior domain knowledge — when AWA* is to be used to satisfy some other bound, such as $C \leq C^* + \log C^*$ for exam-

Algorithm 1 Best-First Search (BFS$^{\Phi}$)

```
1: g(n_i) = 0, pred(n_i) = NONE
2: OPEN ← {n_i}, CLOSED ← {}
3: while OPEN is not empty do
4:    n ← arg min_{n'∈OPEN} Φ(n')
5:    if n is a goal node then
6:       return solution path extracted from CLOSED
7:    for all child n_c of n do
8:       if n_c ∈ OPEN ∪ CLOSED then
9:          if g(n) + c(n, n_c) < g(n_c) then
10:            g(n_c) = g(n) + c(n, n_c)
11:            pred(n_c) ← n
12:            if n_c ∈ CLOSED then
13:               CLOSED← CLOSED−{n_c}
14:               OPEN← OPEN ∪{n_c}
15:      else
16:         g(n_c) = g(n) + c(n, n_c)
17:         pred(n_c) ← n
18:         OPEN ← OPEN ∪{n_c}
19:   CLOSED ← CLOSED ∪{n}
20: return no solution exists
```

ple. This is problematic since this parameter can have a large impact on performance. If AWA* is set to be too greedy, it may find a first solution quickly, but then take too long to improve its incumbent solution to the point of satisfying the bounding function. If AWA* is not set to be greedy enough, it may take too long to find an initial solution. This suggests the need for algorithms specifically targeted towards the given bounding function. Constructing such algorithms is therefore the task considered in the coming sections.

2.3 Bounding in Best-First Search

Best-first search (BFS) is a commonly used search framework that has been used to build many state-of-the-art automated planners for both optimal (Helmert and Röger 2011) and satisficing planning (Richter and Westphal 2010). This popularity is due to its robustness over a wide range of domains. In this section, we will show how to construct BFS-based algorithms for a large class of bounding functions by generalizing the WA* evaluation function.

We define best-first search, pseudocode for which is shown in Figure 1, as a generalization of A* and Djikstra's algorithm, and we assume the reader's familiarity with the use of the OPEN and CLOSED lists in these algorithms. The definition we use generalizes these algorithms by allowing for the use of any arbitrary evaluation function Φ to order nodes on OPEN. Notice that in our generalization we do require that a node on CLOSED is moved back to OPEN whenever a lower cost path is found to it (line 12).

For convenience, we use the notation BFS$^{\Phi}$ to denote a BFS instance which is using the evaluation function Φ. For example, the A* algorithm (Hart, Nilsson, and Raphael 1968) is BFS^{f} where $f(n) = g(n) + h(n)$ and h is admissible, and Dijkstra's algorithm (Dijkstra 1959) is BFS^{g} (Felner 2011). WA* is another instance of BFS (Pohl 1970). It uses the evaluation function $f_\epsilon(n) = g(n) + (1+\epsilon) \cdot h(n)$ (*ie.* WA* is BFS$^{f_\epsilon}$) and is ϵ-admissible if h is admissible. In practice, WA* often solves problems faster than A*. This is a result of f_ϵ which emphasizes the relative importance of h relative to g, and therefore allows WA* to search more greedily on h than does A*.

Also notice that $f_\epsilon(n) = g(n) + B_\epsilon(h(n))$. This suggests the use of the evaluation function $\Phi_B(n) = g(n) + B(h(n))$ for satisfying a given bounding function B since it similarly puts additional emphasis on h. Below, we will show that this approach will suffice for a large family of bounding functions. To do so, we will first require the following lemma:

Lemma 2.1 *Let P_{opt} be any optimal path for a given problem. At any time during a BFS$^{\Phi}$ search prior to a goal node being expanded, there will be a node p on P_{opt} that is on $OPEN$ and such that $g(p) = g^*(p)$.*

This lemma generalizes Lemma 1 of Hart, Nilsson, and Raphael (1968) and it holds for BFS$^{\Phi}$ since the algorithm will move any node on CLOSED back to OPEN whenever a shorter path is found to it. This lemma will now allow us to prove Theorem 2.1 which provides sufficient conditions on Φ for satisfying a given bounding function B.

Theorem 2.2 *Given a bounding function B, BFS^{Φ} will satisfy B if the following conditions hold:*

1. *$\forall$ node n on some optimal path, $\Phi(n) \leq B(g(n)+h^*(n))$*
2. *$\forall$ goal node n_g, $g(n_g) \leq \Phi(n_g)$*

Proof Assume that a goal node n_g has been expanded by BFS$^{\Phi}$. Now consider the search immediately prior to n_g being expanded. By Lemma 2.1, there exists a node p on $OPEN$ such that p is on some optimal path P_{opt} and $g(p) = g^*(p)$. Since n_g is selected for expansion, this implies that $\Phi(p) \geq \Phi(n_g)$ by the definition of BFS. By our assumptions, this means that $B(g(p) + h^*(p)) \geq \Phi(n_g)$. Since $g^*(p) + h^*(p) = C^*$ and $g(p) = g^*(p)$ (since p is on P_{opt}), this also means that $B(C^*) \geq \Phi(n_g)$. Combined with the fact that $g(n_g) \leq \Phi(n_g)$, yields $B(C^*) \geq g(n_g)$. Therefore, BFS$^{\Phi}$ satisfies B. $\square$

So as to concretely demonstrate the implications of Theorem 2.2 consider how it applies to B_ϵ and f_ϵ when h is admissible. For B_ϵ, the first condition on Φ simplifies to $\Phi(n) \leq B_\epsilon(g(n) + h^*(n))$. For any node n on an optimal path, f_ϵ then satisfies this condition since

$$\begin{aligned} f_\epsilon(n) &= g(n) + B_\epsilon(h(n)) && (1)\\ &\leq B_\epsilon(g(n) + h(n)) && (2)\\ &\leq B_\epsilon(g(n) + h^*(n)) && (3) \end{aligned}$$

where line 2 holds because $(1+\epsilon)\cdot(g(n)+h(n)) \geq g(n)+(1+\epsilon)\cdot h(n)$. Since we also have that $g(n_g) = f_\epsilon(n_g)$ for any goal n_g, WA* (*ie.* BFS$^{f_\epsilon}$) satisfies B_ϵ by Theorem 2.2. The following corollary then extends this result to a large class of bounding functions:

Corollary 2.3 *Given bounding function B such that for all x, y, $B(x+y) \geq B(x) + y$, if $\Phi_B(n) = g(n) + B(h(n))$, then BFS$^{\Phi_B}$ with satisfy B.*

This corollary holds because $\Phi_B(n_g) = g(n_g)$ for any goal node n_g, and because the exact same derivation as was performed for f_ϵ applies for Φ_B. This means that for any bounding function B such that for all x, y, $B(x+y) \geq B(x) + y$,

we immediately have an algorithm, specifically BFS$^{\Phi_B}$, for satisfying it. This condition can be viewed as requiring that the maximum difference between C and C^* (*ie.* $C - C^*$) allowed by B cannot be smaller for problems with a large optimal cost than for those with a small optimal cost.

Notice that since $B(x) \geq x$ for all x, $\Phi_B(n) = g(n) + B(h(n))$ will intuitively increase the emphasis on $h(n)$. One exception is in the case of the additive bounding function $B_\gamma(x) = x + \gamma$ (and the resulting evaluation function Φ_{B_γ}), for which nodes will be ordered in OPEN in the same way as in A*. To see this, consider any two nodes n_1 and n_2 such that $\Phi_{B_\gamma}(n_1) \geq \Phi_{B_\gamma}(n_2)$. This means that $g(n_1) + h(n_1) + \gamma \geq g(n_2) + h(n_2) + \gamma$, and so $f(n_1) + \gamma \leq f(n_2) + \gamma$ where $f(n) = g(n) + h(n)$ is the A* evaluation function. This means that $f(n_1) \geq f(n_2)$. As such, the search performed by BFS$^{\Phi}_{B_\gamma}$ will be identical to A*, and so we need to allow for some other way to satisfy B_γ. This is done by the following corollary:

Corollary 2.4 *Given bounding function B such that for all x, y, $B(x + y) \geq B(x) + y$, if Φ is the function $\Phi(n) = g(n) + D(n)$ where h is admissible and $0 \leq D(n) \leq B(h(n))$ for all n, then BFS$^{\Phi}$ will satisfy B.*

This corollary, which follows from an almost identical derivation as Corollary 2.3, introduces a new function D which must be bound by B. For example, consider the following possible D function:

$$D_\gamma(n) = \begin{cases} 0 & \text{if } n \text{ is a goal} \\ h(n) + \gamma & \text{otherwise} \end{cases}$$

By Corollary 2.4 we can use D_γ to satisfy B_γ by constructing the evaluation function $\Phi'(n) = g(n) + D_\gamma(n)$. This function will prioritize goal nodes, though in domains in which the heuristic is correct for all nodes adjacent to a goal node, BFS$^{\Phi'}$ will still be identical to an A* instance which breaks ties in favour of nodes with a low h-cost . However, in Section 4 we will consider other D functions which will successfully trade-off speed for solution quality.

2.4 Bounding in Iterative Deepening Search

Though best-first search remains popular, its memory requirements often limit its effectiveness in large combinatorial domains. In these cases, *Iterative Deepening Depth-First Search* has been shown to be an effective alternative due to its lower memory requirements (Korf 1985). Below, we will show that by using iterative deepening, we can also satisfy a large class of bounding functions.

Iterative deepening depth-first search consists of a sequence of threshold-limited depth-first searches, where some evaluation function Φ is used to define the thresholds. For the first iteration (ie. iteration 0), the threshold t_0 is set as $\Phi(n_i)$. For any other iteration $j > 0$, threshold t_j is set as the smallest Φ-cost of all nodes that were generated but not expanded during iteration $j - 1$. Iterations terminate when either a goal is found or all nodes that satisfy the current threshold have been expanded.

IDA* (Korf 1985) is a special case of this paradigm which uses the A* evaluation function $f(n) = g(n) + h(n)$ where h is admissible. We will let ID$^{\Phi}$ denote the more general version of this algorithm in which any arbitrary evaluation function Φ is being used. For example, IDA$^* =$ ID^f. Just as for best-first search, we first identify sufficient conditions on Φ so that ID$^{\Phi}$ satisfies a given bounding function B. This is done in the following theorem:

Theorem 2.5 *Given a bounding function B, ID$^{\Phi}$ will satisfy B if the following conditions hold:*

1. *$\forall$ node n on an optimal path, $\Phi(n) \leq B(g(n) + h^*(n))$*
2. *$\forall$ goal node n_g, $g(n_g) \leq \Phi(n_g)$*

Proof Assume that a goal node n_g is found on iteration j which uses threshold t_j. This means that $\Phi(n_g) \leq t_j$. Let P_{opt} denote some optimal solution path. As in the proof of Theorem 2.2, we begin by guaranteeing the existence of a node p that is on P_{opt} and for which $\Phi(p) \geq \Phi(n_g)$. To do so, we must consider two cases. If $j = 0$, we can select $p = n_i$ since $\Phi(n_i) = t_0 \geq \Phi(n_g)$, and since n_i is necessarily on any optimal path. If $j > 0$, we will show by contradiction that there exists a node p on P_{opt} for which $\Phi(n) \geq t_j$, and so $\Phi(p) \geq \Phi(n_g)$ since $\Phi(n_g) \leq t_j$. To do so, assume that for any n on P_{opt}, $\Phi(n) < t_j$. However, at least one node n' on P_{opt} must satisfy $\Phi(n') > t_{j-1}$ as otherwise P_{opt} would have been found during iteration $j - 1$. This means that $t_{j-1} < \Phi(n') < t_j$, which contradicts the selection of t_j as the new threshold. Therefore, there is some node p on P_{opt} such that $\Phi(p) \geq t \geq \Phi(n_g)$.

Having guaranteed the existence this node, the proof then proceeds exactly as it did for Theorem 2.2 once a node p on P_{opt} was found for which $\Phi(p) \geq \Phi(n_g)$. □

When comparing Theorems 2.2, we see that the same sufficient conditions hold on Φ for either BFS$^{\Phi}$ or ID$^{\Phi}$. As such, corollaries 2.3 and 2.4 can easily be extended to ID$^{\Phi}$. This means that in practice, when we want to satisfy a bounding function B, we simply need to find an evaluation function Φ that satisfies the properties in these theorems and then decide based on domain properties (such as state-space size, or number of cycles) on whether to use BFS$^{\Phi}$ or ID$^{\Phi}$.

2.5 Bounding in Focal List Search

In recent years, focal list based search has been shown to be an effective alternative to best-first search in domains with non-uniform edge costs. In this section, we generalize this class of algorithms so as to be able to satisfy any monotonically non-decreasing bounding function in such domains.

We begin by examining the first algorithm of this kind: A$^*_\epsilon$ (Pearl and Kim 1982). A$^*_\epsilon$ is similar to BFS, except it replaces line 4 in Algorithm 1 with a two-step process. In the first step, a data structure called the *focal list* is constructed so as to contain the following subset of OPEN:

$$\text{FOCAL} = \{n | f(n) \leq (1 + \epsilon) \cdot \min_{n' \in OPEN} f(n')\}$$

where $f(n) = g(n) + h(n)$ is the A* evaluation function, and h is admissible. This means that FOCAL contains all nodes with an f-cost no more than a factor of $(1+\epsilon)$ greater than the node with the lowest f-cost. In the second step, a node n is selected for expansion from FOCAL as follows:

$$n \leftarrow \arg \min_{n' \in \text{FOCAL}} h_2(n')$$

where h_2 is some secondary heuristic. Typically, h_2 will estimate the *distance-to-go* of a node, which can be understood as estimating the length of the shortest path from the node to the goal (in terms of number of actions), unlike h which will estimates the cost of that path and is thus a *cost-to-go* heuristic (Thayer and Ruml 2011). When a distance-to-go heuristic is not available, one can simply set $h_2 = h$ (Pearl and Kim 1982). Moreover, A^*_ϵ may use any policy for selecting nodes from FOCAL and still remain an ϵ-admissible algorithm (Ebendt and Drechsler 2009). Such is the approach taken by algorithms such as EES (Thayer and Ruml 2011).

Now consider how the size of FOCAL changes with different values of ϵ. When $\epsilon = 0$, FOCAL will only contain those nodes whose f-cost is equal with the node on OPEN with the minimum f-cost. In these cases, h_2 is only used for tie-breaking. However, if ϵ is large, then FOCAL will contain a larger set of nodes with a variety of f-costs, and thus will be allowed to explore them more greedily according to h_2. For example, when $\epsilon = \infty$, $FOCAL = OPEN$ and the search simply expands greedily according to h_2.

A similar behaviour occurs in our generalization of A^*_ϵ for a particular monotonically non-decreasing bounding function B. For this generalization we define FOCAL as follows:

$$\text{FOCAL} = \{n | f(n) \leq B(\min_{n' \in OPEN} f(n'))\}$$

where $f(n) = g(n)+h(n)$ and h is admissible. This too will result in larger focal lists as the bounding function allows for more suboptimality. We will refer to a focal list based algorithm which builds FOCAL in this way as FSEARCH^B, regardless of the policy it uses to select nodes from FOCAL for expansion. In the following theorem, we show that this algorithm also satisfies B:

Theorem 2.6 *Given any monotonically non-decreasing bounding function B, FSEARCH^B will satisfy B.*

Proof Assume that FSEARCH^B has expanded a goal node n_g, and consider the search immediately prior to the expansion of n_g. When n_g is expanded, it is on FOCAL which means that $f(n_g) \leq B(\min_{n' \in \text{OPEN}} f(n'))$. As Lemma 2.1 can easily be extended to apply to focal list based search, we are guaranteed that there is a node p in $OPEN$ such that $g(p) = g^*(p)$ and p is on an optimal path.

By definition, $\min_{n' \in \text{OPEN}} f(n') \leq f(p)$ and so $B(\min_{n' \in \text{OPEN}} f(n')) \leq B(f(p))$ since B is monotonically non-decreasing. This means that $f(n_g) \leq B(f(p))$. Since $g(p) = g^*(p)$ and h is admissible, $f(p) \leq g^*(p) + h^*(p) = C^*$. Combined with the fact that $f(n_g) = g(n_g)$ this means that $g(n_g) \leq B(C^*)$. □

Again, notice that this proof holds regardless of the policy used to select nodes from FOCAL for expansion, and so we can construct versions of A^*_ϵ and EES so as to satisfy any monotonically non-decreasing bounding function. We will experiment with these in the additive setting in Section 4.3.

3 Evaluation Functions for B_γ

As described in Section 2.3, unlike most bounding functions, the evaluation function suggested by Corollary 2.3 (*ie.* Φ_B) will be ineffective when used for satisfying $B_\gamma(x) = x + \gamma$. In this section, we use Corollary 2.4 to construct other bounding functions to remedy this problem. This means that we will construct evaluation functions of the type $\Phi(n) = g(n) + D(n)$ where $0 \leq D(n) \leq h(n) + \gamma$. We consider two possible cases: when we are given only an admissible function h, and when we have both an admissible function h_a and an inadmissible function h_i.

3.1 Using Admissible Heuristics

Assume that we are given an admissible function h for search. The guiding principle which we will use for constructing evaluation functions so as to satisfy B_γ will be to follow the example of the WA* evaluation function and further emphasize the role of the heuristic. This is the idea behind our next evaluation function, which penalizes nodes with a high heuristic value by adding a term that is linear in the heuristic. The key difference between the new function and the WA* function f_ϵ is that in order to satisfy B_γ, this penalty must be guaranteed to be no greater than γ. In this end, we consider the following function:

$$F'(n) = g(n) + h(n) + \frac{h(n)}{h_{max}} \cdot \gamma$$

where h_{max} is a constant such that for all n, $h(n) \leq h_{max}$. This condition on h_{max} guarantees that the corresponding D-function, given by $D(n) = h(n)+h(n)\cdot\gamma/h_{max}$ satisfies the required relation that $0 \leq D(n) \leq h(n) + \gamma$ for all n.

Also notice that this evaluation function is equivalent to f_ϵ where $\epsilon = \gamma/h_{max}$. This means that if h_{max} is a loose upper bound and we were using best-first search, then our algorithm would be equivalent to a WA* search that is not using as high of a weight as it could. Using a tight upper bound on h for h_{max} is therefore crucial for achieving good performance. Unfortunately, a tight upper bound on the heuristic value of any state may not be immediately available without extensive domain knowledge. Moreover, even such an upper bound may not be relevant for a given particular starting state, as it may over-estimate the heuristic values that will actually be seen during search. This leads us to consider the following evaluation function:

$$F_\gamma(n) = g(n) + h(n) + \frac{\min(h(n), h(n_i))}{h(n_i)} \cdot \gamma$$

where n_i is the initial node. Intuitively, this function uses $h(n_i)$ as a more instance relevant upper bound and penalizes nodes according to how much *heuristic progress* has been made, where progress is measured by $h(n)/h(n_i)$.

Notice that if $h(n_i)$ is the largest heuristic value seen during the search, then the 'min' can be removed from F_γ which then becomes equivalent to f_ϵ in which $\epsilon = (1 + \gamma/h(n_i))$. In general, the 'min' cannot be omitted as it is necessary for the additive bound, making it the crucial difference between F_γ and f_ϵ. However, omitting the 'min' gives us insight into what to expect from F_γ and F'. Specifically, since F_γ corresponds to f_ϵ with $\epsilon = (1 + \gamma/h(n_i))$ and F' corresponds to f'_ϵ with $\epsilon' = (1+\gamma/h_{max})$, we expect that when comparing the two, BFS$^{F_\gamma}$ will be greedier (and therefore usually faster) since $h(n_i) \leq h_{max}$.

We verified this with experiments on the 15-puzzle using the 7-8 additive pattern database heuristic (Felner, Hanan, and Korf 2011). For every γ in the set 2, 4, 8, ..., 256, the average number of nodes expanded by BFS$^{F_\gamma}$ was less than was expanded by BFS$^{F'}$. As such, in all further experiments with BFS and ID, we will use F_γ instead of F'.

3.2 Using Inadmissible Heuristics in BFS and ID

While Theorems 2.2 and 2.5 offer a general way to construct evaluation functions for arbitrary bounding functions, an alternative is to come up with an inadmissible heuristic with a *bounded amount of inadmissibility*. This is possible due to the following corollary:

Corollary 3.1 *Given a bounding function B such that for all x, y, $B(x + y) \geq B(x) + y$, if $\Phi(n) = g(n) + h(n)$ where $0 \leq h(n) \leq B(h^*(n))$ for all n, then BFS$^\Phi$ and ID$^\Phi$ with satisfy B.*

This corollary, which follows from an almost identical proof as was given for corollary 2.3, generalizes a theorem by Harris (1974) which was specific to the case of additive bounds.

For most modern-day inadmissible heuristics, such as the FF heuristic (Hoffmann and Nebel 2001) and the bootstrap heuristic (Jabbari Arfaee, Zilles, and Holte 2011), there are no known bounds on heuristic inadmissibility. However, we can still employ an inadmissible heuristic h_i in conjunction with an admissible heuristic h_a, by forcing a limit on the difference between the two. In the case of additive bounding, this means that we will use the following function:

$$IL{-}h_i(n) = g(n) + \min(h_a(n) + \gamma, h_i(n))$$

Note, $IL{-}h_i$ stands for inadmissible limiting of h_i. This evaluation function allows us to exploit the inadmissible heuristic as much as possible unless the difference between h_a and h_i is too large. When they do disagree by too much (*ie.* $h_i(n) > h_a(n)+\gamma$) we cannot simply use $h_i(n)$ and still be guaranteed to satisfy the bound. As such, we use the value of $h_a(n) + \gamma$ since this penalizes the node (by increasing its evaluation) by as much as possible while still satisfying B_γ.

4 Experiments

In this section, we demonstrate that the theory developed in Section 2 and the evaluation functions developed in Section 3 can be used to construct algorithms that effectively trade-off runtime and guaranteed solution quality. This is shown for the additive bounding function $B_\gamma(x) = x+\gamma$ for a range of γ values. Note, we do not compare against ϵ-admissible algorithms that do not have an anytime component for satisfying a given additive bound γ. This is because it is not clear how to parameterize these algorithms so as to guarantee that they satisfy B_γ without prior knowledge about the problem. In the next section we will also show that this issue also effects AWA* when it is used for additive bounding, as the algorithm's effectiveness, relative to the use of an additive BFS, will depend on how well it was parameterized.

4.1 Additive Best-First Search

In this section, we consider the use of BFS to satisfy B_γ. In particular, we will show that additive BFS will allow us to effectively trade-off guaranteed solution suboptimality for coverage. We begin by comparing BFS$^{F_\gamma}$ (where F_γ was described in Section 3) against AWA* as used for additive bounding (described in 2.2). We then show that additive BFS can also be used effectively for automated planning.

		Additive Suboptimality Bound / γ								
	w	0	2	4	8	16	32	64	128	256
AWA*	1.5	**3,013**	2,850	2,180	606	**117**	99	99	99	99
AWA*	2.0	4,614	4,553	4,222	2,330	343	53	53	53	53
AWA*	2.5	5,248	5,215	4,979	3,185	678	**49**	**40**	40	40
BFS$^{F_\gamma}$		3,732	**2,460**	**1,175**	**403**	**117**	66	44	**33**	**30**

Table 2: The average number of expanded nodes (in tens of nodes) by Bounded AWA* and BFS$^{F_\gamma}$ in the 15-puzzle.

Comparing BFS$^{F_\gamma}$ and AWA* To compare BFS$^{F_\gamma}$ against AWA* for additive bounding, we ran the two methods on the 100 15-puzzle problems given by Korf (1985) with different additive suboptimality bounds. The results of this experiment are shown in Table 2, which shows the average number of node expansions needed (in tens of nodes) to satisfy the desired bound. For every bound, the algorithm that expanded the fewest nodes is marked in bold. AWA* was tested with several different weights (w) since it is not obvious how to parameterize it for a specific γ.

The table shows that BFS$^{F_\gamma}$ exhibits the desired tradeoff of suboptimality and time as it expands fewer nodes as the suboptimality bound is loosened (recall that BFS$^{F_\gamma}$ is equivalent to A* when $\gamma = 0$). The general trend when using AWA* set with a particular weight is that it works well over some range of γ values, but no single weight gets strong performance everywhere. This is unlike BFS$^{F_\gamma}$ which exhibits strong performance over all tested values of γ. BFS^{F_γ} also has the additional advantage over AWA* in that it does not need to maintain two separate open lists (the second being used to compute the lower bound on C^*). This overhead meant that even in the cases in which AWA* expanded fewer nodes, it still had a longer runtime than BFS^{F_γ}.

In conclusion, running AWA* with an additive bound suffers from two issues: it is unclear as to how to initially select a weight for a particular bound, and the computational expense of maintaining two open lists. As weight selection will be even more difficult in an automated planner (since less is known about the domains), we restrict our experiments in planning domains to the additive BFS algorithms.

Additive BFS for Planning We will now show that additive BFS algorithms can also be effective in automated planners. For these experiments, we implemented F_γ and inadmissibility limiting in the Fast Downward (Helmert 2006) framework. The admissible heuristic used is LM-Cut (Helmert and Domshlak 2009), while the inadmissible FF heuristic (Hoffmann and Nebel 2001) is used when using inadmissibility limiting. On each problem, configurations were given a 30 minute time and 4 GB memory limit when running on a machine with two 2.19 GHz AMD Opteron 248 processors. For the test suite, we used the 280 problems from the optimal track of IPC 2011 with action costs ignored. This

	Additive Suboptimality Bound / γ									
	0	1	2	5	10	25	50	100	500	1000
BFS$^{F_\gamma}$	132	**145**	**148**	**163**	**175**	191	204	**216**	**218**	**218**
BFS^{IL-FF}	NA	142	146	157	172	199	199	199	199	199
P-BFS^{IL-FF}	NA	142	146	159	**175**	**213**	**214**	214	214	214

Table 3: The coverage of additive BFS algorithms in planning domains.

means that all tasks are treated as unit cost tasks. This was done so as to determine if BFS shows similar behaviour in the additive setting as it does for ϵ-admissible search in the situations in which it is known to be most effective. Typically, non-unit cost tasks are best handled by focal list based approaches (Thayer and Ruml 2011), as will be verified in the additive setting in Section 4.3.

Let us first consider the coverage of BFS$^{F_\gamma}$ on these planning domains, which is shown in the first row of Table 3. The table shows that this algorithm exhibits the desired behaviour: as suboptimality is allowed to increase, so too does the coverage. This is consistent with the behaviour of WA*, which also benefits from the additional greediness allowed with a looser bound. The second row of Table 3 then shows the performance when the inadmissibility of the FF heuristic is limited using LM-Cut. Again we see that as the amount of suboptimality is increased, so too does the coverage. However, the coverage generally lags behind that of BFS$^{F_\gamma}$. Subsequent analysis showed that this was partially caused by the overhead of calculating two heuristic values for each node. To combat this effect, we used a *pathmax*-like technique (Mero 1984) with which the LM-cut heuristic can be computed less frequently. To do so, notice that for any node n, $h^*(n') \geq h^*(n) - c(n,n)$ where n' is a child of n. Therefore, for any admissible h_a and inadmissible h_i, if $h_i(n') \leq h_a(n) - c(n,n) + \gamma$, we can infer that $h_i(n') \leq h^*(n') + \gamma$ without calculating $h_a(n')$ by Corollary 2.4. We can similarly avoid calculating $h_a(n'')$ for any successor n'' of n for which $h_i(n'') \leq h_a(n'') - C_{n,n''}$ where $C_{n,n''}$ is the cost of the path from n to n''.

The coverage seen when this technique is added to BFS^{IL-FF} is shown in the third row of Table 3 (denoted as P-BFS^{IL-FF}). The pathmax technique clearly offers significant gains, and now inadmissibility limiting is equal or better than BFS$^{F_\gamma}$ for $10 \leq \gamma < 100$. For low values of γ, P-BFS^{IL-FF} suffers because the inadmissible heuristic is limited too often. To illustrate this effect, consider what happens when the inadmissible heuristic is always limited. This means that $\min(h_n(c) + \gamma, h_i(n)) = h_a(n) + \gamma$ for all n. When this happens, IL-FF degenerates to $g(n) + h_a(n) + \gamma$, which, as discussed in Section 2.3 will result in a search that is identical to A^*. As γ gets smaller, the frequency with which h_i is limited increases and so we approach this case.

When γ gets large enough, the inadmissible heuristic will never be limited and IL-FF degenerates to $g(n) + h_i(n)$. We can see this happening in Table 3, as P-BFS^{IL-FF} stops increasing in coverage at $\gamma = 50$. We confirmed this as the cause by using the FF heuristic without limiting, in which case BFS solves 218 problems. This small discrepancy in coverage is caused by the overhead of using LM-Cut.

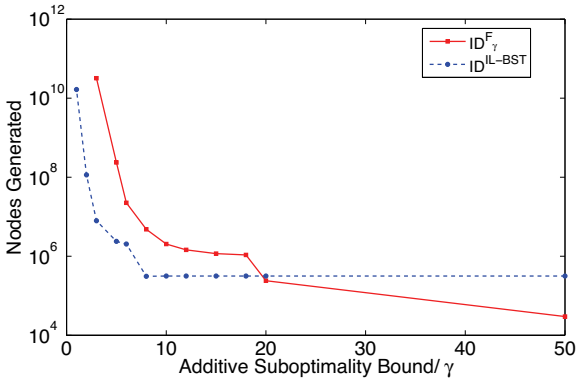

Figure 1: Additive BFS in the 15-blocks world.

Though P-BFS^{IL-FF} has a lower coverage than BFS$^{F_\gamma}$ for low and high values of γ, it still sits on the Pareto-optimal front when comparing these two algorithms, and it does show that we can use inadmissible heuristics in BFS and still have guaranteed bounds. We will see similar results below when evaluating additive iterative deepening algorithms.

4.2 Additive Iterative Deepening Search

This paradigm is typically used on large combinatorial state-spaces in which the memory requirements of best-first search can limit its effectiveness. We thus test in two such domains, the 24-puzzle and 15-blocks world, and see similar behaviour as with the additive BFS algorithms. In the 24-puzzle, the admissible heuristic used is the 6-6-6-6 additive heuristic developed by Korf and Felner (2002). In the 15-blocks world problem, the admissible heuristic is given by the number of out-of-place blocks. In both domains, the inadmissible heuristic used for inadmissibility limiting is the bootstrap learned heuristic (Jabbari Arfaee, Zilles, and Holte 2011). This is a machine learned heuristic that is constructed in an offline training phase. Due to its machine learned nature, there are no known bounds on its inadmissibility.

The results for the blocks world experiment is shown in Figure 1 (notice the log scale on the vertical axis), in which $IL{-}BST$ denotes the inadmissibility limiting evaluation function when using the bootstrap learning heuristic. The general trend for ID$^{F_\gamma}$ and ID^{IL-BST} is that they both improve their runtime as the bound increases, though ID^{IL-BST} stops improving when the inadmissible heuristic is no longer ever limited. This happens at $\gamma = 9$ in this domain due to the relatively high accuracy of the admissible heuristic. Because ID$^{F_\gamma}$ can continue to become greedy on h_a, it is able to surpass the performance of ID^{IL-BST} for $\gamma \geq 20$. Similar behaviour is seen in the 24-puzzle (details omitted due to space constraints) in which the performance of ID$^{F_\gamma}$ stops improving at $\gamma = 25$, which is the same point at which ID$^{F_\gamma}$ becomes the faster algorithm. However, aside from small values of γ in which the inadmissible heuristic is limited too often, $IL{-}BST$ outperforms F_γ for an intermediate γ range. This means that just as with BFS, the theory in Section 2 led to the successful construction of additive iterative deepening algorithms.

4.3 Additive Focal List Search

Focal list based search has been shown to be more effective than BFS in domains with non-uniform action costs. In this section, we will demonstrate that the same is true for the additive versions of these algorithms. We begin by considering the additive version of A^*_ϵ. As described by Theorem 2.6, the suboptimality of A^*_ϵ can be adjusted by changing the criteria for including a node on the OPEN list to the following:

$$\text{FOCAL} = \{n | f(n) \leq \min_{n' \in OPEN} f(n') + \gamma\}$$

Once that is done, whatever heuristic was to be used to select from FOCAL in A^*_ϵ can still be used in the additive version while still satisfying the bound.

Explicit Estimation Search (EES) is a newer focal list based algorithm that has been shown to outperform A^*_ϵ in many domains (Thayer and Ruml 2011). It uses a more sophisticated policy for selecting nodes from FOCAL for expansion. We will not describe this policy further here (see (Thayer and Ruml 2011) for details), except to mention that it can also be changed into an additively bounded algorithm by changing the way that FOCAL is defined. This additive version of EES (referred to as FSEARCH$^{B_\gamma}$-EES since it is an instance of FSEARCH$^{B_\gamma}$ which selects nodes from FOCAL according to the EES policy) was then tested against BFS$^{F_\gamma}$ and the additive version of A^*_ϵ (referred to as FSEARCH$^{B_\gamma}$-A^*_ϵ) on a variant of the 15-puzzle in which the cost of moving a tile is given by the inverse of the tile's face value. The admissible heuristic used is a cost-based version of Manhattan distance, while standard Manhattan distance is used as the distance-to-go heuristic.

Figure 2 shows the performance of these three algorithms on this domain. Notice that with the introduction of action costs, BFS$^{F_\gamma}$ is not exchanging guaranteed suboptimality for speed and it is much weaker than the focal list based algorithms. In addition, we see that FSEARCH$^{B_\gamma}$-EES outperforms FSEARCH$^{B_\gamma}$-A^*_ϵ for low values of γ, while FSEARCH$^{B_\gamma}$-A^*_ϵ shows a slight advantage for large γ. These results, including the poor performance of BFS in this domain, are consistent with those seen when using the ϵ-admissible versions of these algorithms in this domain. We also experimented in the dynamic robot navigation and the heavy vacuum domains, in which FSEARCH$^{B_\gamma}$-EES substantially outperforms FSEARCH$^{B_\gamma}$-A^*_ϵ, just as EES outperforms A^*_ϵ when using ϵ-admissible bounding (Thayer and Ruml 2011). That is, these originally ϵ-admissible algorithms have retained their relative strengths and weaknesses when modified so as to satisfy a different bounding function.

5 Related Work

Dechter and Pearl (1985) previously provided bounds on the cost of a solution found when using a best-first search guided by alternative evaluation functions. Further generalizations were considered by Farreny (1999) who also looked at the use of alternative focal list definitions in focal list based algorithms. The goal of our paper can be seen as the inverse of these papers in that the works of Dechter and Pearl, and by Farreny, attempt to determine the suboptimality of solutions found by a given algorithm. In contrast, our goal is to construct algorithms for a given bounding paradigm.

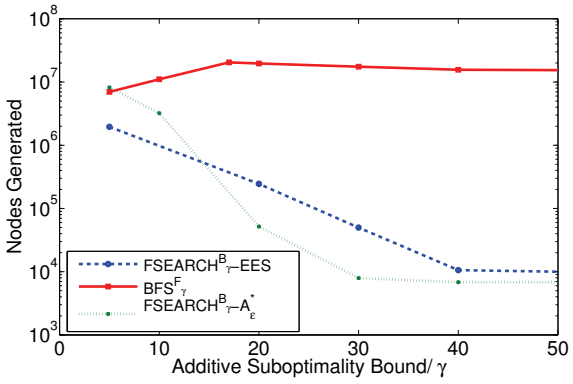

Figure 2: Additive focal list based search in the inverse 15-puzzle.

An alternative form of bounding was also investigated by Stern, Puzis, and Felner (2011). In this paradigm, the cost of any solution found must be no larger than some given constant K, regardless of what the optimal solution cost is. This paradigm fits into our functional view of bounding, specifically as $B_K(x) = K$ when $K \geq C^*$. Our theory does not offer an immediate way to construct BFS algorithms for this paradigm since it is not true that $\forall x, y, B_K(x + y) \geq B_K(x) + y$. However, it has been shown that focal list based algorithms of the type suggested by Theorem 2.6 are effective for this bounding paradigm (Thayer et al. 2012).

Additive bounding has previously been considered by Harris (1974) who showed that if A* was used with a heuristic h for which $h(n) \leq h^*(n)+\gamma$ for all nodes n, then the solution found will have a cost no more than $C^* + \gamma$. This condition was generalized in Section 3.2. More recently, it was shown that a particular termination criteria induced an additive bound when using bi-directional search (Rice and Tsotras 2012). We consider generalizing bi-directional search so as to satisfy other bounding functions as future work.

6 Conclusion

In this paper, we have generalized the notion of a suboptimality bound using a functional definition that allows for the use of alternate suboptimality measures. We then developed theory which showed that four different search frameworks — anytime, best-first search, iterative deepening, and focal list based algorithms — could be modified so as to satisfy a large set of suboptimality bounds. This allows a system designer to not only measure and bound suboptimality as they see fit, but to then satisfy that bound while selecting the most appropriate framework for their particular domain. We then showed that the theory suggests practical algorithms by using it to construct algorithms with additive bounds which efficiently trade-off suboptimality for runtime.

7 Acknowledgments

We would like to thank Jonathan Schaeffer and Ariel Felner for their advice on this paper. This research was supported by GRAND.

References

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

Dijkstra, E. W. 1959. A note on two problems in connexion with graphs. *Numerische Mathematik* 1:269–271.

Ebendt, R., and Drechsler, R. 2009. Weighted A* search - unifying view and application. *Artificial Intelligence* 173(14):1310–1342.

Farreny, H. 1999. Completeness and Admissibility for General Heuristic Search Algorithms-A Theoretical Study: Basic Concepts and Proofs. *J. Heuristics* 5(3):353–376.

Felner, A.; Hanan, S.; and Korf, R. E. 2011. Additive Pattern Database Heuristics. *CoRR* abs/1107.0050.

Felner, A. 2011. Position Paper: Dijkstra's Algorithm versus Uniform Cost Search or a Case Against Dijkstra's Algorithm. In *SOCS*.

Hansen, E. A., and Zhou, R. 2007. Anytime Heuristic Search. *JAIR* 28:267–297.

Harris, L. R. 1974. The Heuristic Search under Conditions of Error. *Artificial Intelligence* 5(3):217–234.

Hart, P. E.; Nilsson, N. J.; and 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.

Helmert, M., and Domshlak, C. 2009. Landmarks, Critical Paths and Abstractions: What's the Difference Anyway? In *ICAPS*.

Helmert, M., and Röger, G. 2011. Fast Downward Stone Soup: A Baseline for Building Planner Portfolios. In *ICAPS-2011 Workshop on Planning and Learning*, 28–35.

Helmert, M. 2006. The Fast Downward Planning System. *JAIR* 26:191–246.

Hoffmann, J., and Nebel, B. 2001. The FF Planning System: Fast Plan Generation Through Heuristic Search. *JAIR* 14:253–302.

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

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

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

Mero, L. 1984. A Heuristic Search Algorithm with Modifiable Estimate. *Artificial Intelligence* 23:13–27.

Pearl, J., and Kim, J. H. 1982. Studies in Semi-Admissible Heuristics. *IEEE Trans. on Pattern Recognition and Machine Intelligence* 4(4):392–399.

Pohl, I. 1970. Heuristic search viewed as path finding in a graph. *Artificial Intelligence* 1(3-4):193–204.

Rice, M. N., and Tsotras, V. J. 2012. Bidirectional A* Search with Additive Approximation Bounds. In *SOCS*.

Richter, S., and Westphal, M. 2010. The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks. *JAIR* 39:127–177.

Stern, R. T.; Puzis, R.; and Felner, A. 2011. Potential Search: A Bounded-Cost Search Algorithm. In *ICAPS*.

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. 2011. Bounded Suboptimal Search: A Direct Approach Using Inadmissible Estimates. In *IJCAI*, 674–679.

Thayer, J. T.; Stern, R.; Felner, A.; and Ruml, W. 2012. Faster Bounded-Cost Search Using Inadmissible Estimates. In *ICAPS*.