# Anytime Focal Search with Applications

**Liron Cohen[1], Matias Greco[2], Hang Ma[1],**
**Carlos Hernandez[2], Ariel Felner[3], T. K. Satish Kumar[1] and Sven Koenig[1]**
[1]University of Southern California
[2]Universidad Andres Bello
[3]Ben-Gurion University

## Abstract

Focal search (FS) is a bounded-suboptimal search (BSS) variant of A*. Like A*, it uses an open list whose states are sorted in increasing order of their $f$-values. Unlike A*, it also uses a focal list containing all states from the open list whose $f$-values are no larger than a suboptimality factor times the smallest $f$-value in the open list. In this paper, we develop an anytime version of FS, called anytime FS (AFS), that is useful when deliberation time is limited. AFS finds a "good" solution quickly and refines it to better and better solutions if time allows. It does this refinement efficiently by reusing previous search efforts. On the theoretical side, we show that AFS is bounded suboptimal and that anytime potential search (ATPS/ANA*), a state-of-the-art anytime bounded-cost search (BCS) variant of A*, is a special case of AFS. In doing so, we bridge the gap between anytime search algorithms based on BSS and BCS. We also identify different properties of priority functions, used to sort the focal list, that may allow for efficient reuse of previous search efforts. On the experimental side, we demonstrate the usefulness of AFS for solving hard combinatorial problems, such as the generalized covering traveling salesman problem and the multi-agent pathfinding problem.

## 1 Introduction

A* [Hart *et al.*, 1968] is a best-first search algorithm that continuously expands a state with minimal key from OPEN[1], where the key of state $n$ is given by $f(n) = g(n) + h(n)$. Here, $g(n)$ is the distance of $n$ from the start state computed and maintained by A*, and $h(n)$ is the state's cost-to-goal estimate (heuristic value). Despite the many successes of A*, it is known to be unviable for large combinatorial problems when heuristic guidance is not perfect [Helmert and Roger, 2008]. This has prompted the development of several variants of A* that have the freedom to produce suboptimal solutions since this freedom often leads to faster runtimes [Wilt and Ruml, 2016]. Nevertheless, in many real-world domains, such as in robotics and probabilistic reasoning, solution cost cannot be compromised beyond a reasonable factor.

Hence, subsequent work has focused on bounded-suboptimal search (BSS), that tries to trade-off solution cost with runtime. BSS algorithms produce solutions with costs at most $w$ times the optimal cost, for some user-specified suboptimality bound $w \geq 1$. One such algorithm is weighted-A* ($w$A*) [Pohl, 1970]. $w$A* differs from A* only in the keys it uses: It puts more weight on the heuristic value by inflating it by an inflation factor $w$, that is, $f(n) = g(n) + wh(n)$. $w$A* generates solutions faster than A* in many domains [Bonet and Geffner, 2001; Korf, 1993]. However, increasing the weight of the heuristic value may also lead to larger runtimes, especially when the correlation between the heuristic values and the minimal number of edges-to-goal is weak [Wilt and Ruml, 2012].

Inflating heuristic values also allows for the development of anytime search algorithms [Thayer and Ruml, 2010]. Anytime algorithms are useful when deliberation time is limited. They are intended to generate an initial solution quickly and use any additional available time to generate better and better solutions. ARA* [Likhachev *et al.*, 2003] is an anytime heuristic search algorithm that repeatedly runs $w$A* with decreasing values of $w$. ARA* reuses search efforts from previous search iterations and is considered efficient since it expands each state at most once per search iteration. This efficiency property relies on bounded admissibility[2] [Aine and Likhachev, 2016].

Since ARA* is based on $w$A*, it is subject to a restric-

Contact author: lironcoh@usc.edu

The research at the University of Southern California was supported by the National Science Foundation (NSF) under grant numbers 1724392, 1409987, and 1319966 as well as a gift from Amazon. Matias Greco and Carlos Hernandez were partially funded by Fondecyt grant number 1161526. Ariel Felner was funded by the Israel Science Foundation grant number 844/17.

[1]OPEN is the set of generated and not expanded states.

[2]A state $n$ is said to be *bounded admissible* iff $g(n) \leq wg^*(n)$ when it is selected for expansion, where $g^*(n)$ is the distance from the start state to $n$.

tion: Like A*, it expands states greedily in order of increasing $f$-values from OPEN. Therefore, its intended trade-off between solution quality and runtime stems from the inflation of heuristic values rather than the freedom of expanding states with suboptimal $f$-values. Unlike A*, $w$A* or ARA*, focal search (FS) [Pearl and Kim, 1982] leverages this freedom to expand states with suboptimal $f$-values. FS guarantees bounded suboptimality by using $f$-values in conjunction with arbitrary priorities to order state expansions. While the $f$-values determine a set of possible states (denoted FOCAL) that qualify for expansion, the arbitrary priorities are used to choose a particular state for expansion from FOCAL. FS has been successfully used to efficiently solve many combinatorial problems [Hatem and Ruml, 2014; Barer *et al.*, 2014].

In this paper, we therefore develop an anytime version of FS, called anytime FS (AFS). Because the source of suboptimality in FS comes from the *flexibility* of expanding states with suboptimal $f$-values rather than the *inflation* of $h$-values, AFS works by iteratively tightening the flexibility rather than adopting ARA*'s strategy of iteratively decreasing the inflation factor. Like ARA*, AFS also reuses search efforts from previous search iterations while guaranteeing the suboptimality bounds. In addition, the mechanism that AFS uses to update FOCAL between consecutive search iterations is easy to implement and analyze.

For pedagogical reasons, we also relate our work to the bounded-cost search (BCS) framework and its anytime adaptations. In BCS, a cost bound $C$ is given and the task is to find a solution with cost at most $C$ as fast as possible. Two state-of-the-art anytime BCS algorithms, anytime potential search (ATPS) [Stern *et al.*, 2011] and anytime non-parametric A* (ANA*) [van den Berg *et al.*, 2011], have been shown to be equivalent [Stern *et al.*, 2014]. Both ATPS and ANA* can be thought of as AFS that uses a specific mechanism for iteratively tightening FOCAL and a specific priority function, called the potential function, to sort it. Furthermore, Gilon et al. 2016 have recently shown that any BCS algorithm can be transformed to the BSS framework and vice versa.

On the theoretical side, we show the bounded suboptimality of AFS, identify different ways to define FOCAL along with properties of priority functions used to sort it, and thus bridge the gap between anytime BSS and anytime BCS. On the experimental side, we demonstrate the usefulness of AFS for solving hard combinatorial problems, such as the generalized covering traveling salesman problem and the multi-agent pathfinding problem.

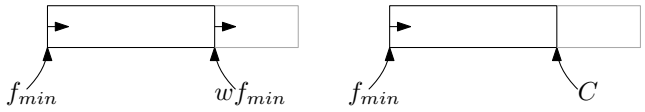
## 2 Focal Search (FS)

Two prominent suboptimal search frameworks, BSS and BCS, are defined as follow: Given a user-specified suboptimality bound $w \geq 1$, a BSS algorithm is guaranteed to find a solution of cost at most $wP_{opt}$, where $P_{opt}$ is the cost of an optimal solution. Given a user-specified cost bound $C \geq 0$, a BCS algorithm is guaranteed to find a solution of cost at most $C$.

We now characterize FS under both frameworks and develop a unified view. FS has two main components that are independent of each other. The first one is about which states are in FOCAL, and the second one is about which priority function is used to sort FOCAL.

| | |
|---|---|
| BSS | Bounded-suboptimal search |
| BCS | Bounded-cost search |
| FS | Focal search |
| PS | Potential search |
| AFS | Anytime focal search |
| ATPS | Anytime potential search |
| ANA* | Anytime non-parametric A* |
| $w$A* | Weighted A* |
| ARA* | Anytime repairing A* |
| EES | Explicit estimation search |
| GCTSP | Generalized cost traveling salesman problem |
| MAPF | Multi-agent pathfinding |

Table 1: Acronyms




(a) BSS's FOCAL and OPEN (b) BCS's FOCAL and OPEN

Figure 1: Illustrates FOCAL (black) and OPEN (black+grey) for BSS and BCS in (a) and (b), respectively.

### 2.1 Focal List

We use OPEN to denote A*'s open list, which is sorted in increasing order of $f(n) = g(n) + h(n)$, where $h$ is consistent. We also define $f_{min} = \min_{n \in \text{OPEN}} f(n)$ and $\texttt{head}(\text{OPEN}) - \arg\min_{n \in \text{OPEN}} f(n)$.

**Definition 1** (Focal list (FOCAL))**.** There are two ways to define FOCAL $\subseteq$ OPEN:

1. FOCAL $= \{n \in \text{OPEN} : f(n) \leq wf_{min}\}$ for a user-specified suboptimality bound $w \geq 1$.
2. FOCAL $= \{n \in \text{OPEN} : f(n) \leq C\}$ for a user-specified cost bound $C \geq 0$.

FS in the BSS framework is based on the following observation: While A* with admissible heuristic values might spend a long time identifying the best solution among many "good" solutions by expanding only states whose $f$-values equal $f_{min}$, FS has the freedom to choose any "good enough" solution by expanding any state from FOCAL given in Definition 1(1). This flexibility allows FS to terminate earlier than A* while providing bounded suboptimality guarantees.

FOCAL is also useful in the BCS framework. Here, the largest $f$-value in FOCAL does not depend on $f_{min}$ . Instead, we are given a cost bound and the task is to find a solution as fast as possible whose cost is no greater than this cost bound. Once again, we are free to expand any state from FOCAL given in Definition 1(2)[3] and are not constrained to states with minimum $f$-values only.

Figure 1 illustrates FOCAL in the BSS and BCS frameworks. In both frameworks, $f_{min}$ represents smallest $f$-value

[3]We note that it is common to not maintain OPEN explicitly in the BCS framework because $f_{min}$ does not play a role in its FOCAL. This has implications on its suboptimality bound, as we discuss in the context of Lemma 2.

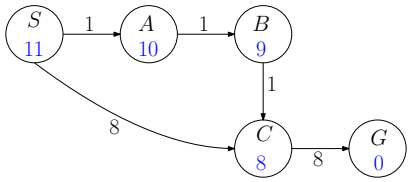



Figure 2: Shows an $h_{\text{FOCAL}}$ that is not $w$-admissible.

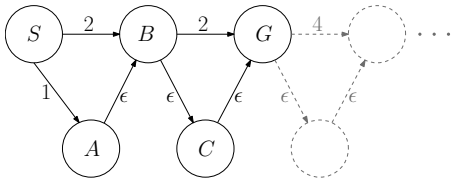



Figure 3: Shows that FS may require re-expansions to guarantee bounded-suboptimality.

of all states in FOCAL. The difference between the two focal lists is in the largest $f$-value of all states in them. In (a), the largest such f-value increases when $f_{min}$ increases (depicted by the right arrow), while, in (b), it remains fixed throughout the search.

## 2.2 Priority Function

The freedom to expand any state in FOCAL allows FS to find a suboptimal solution and terminate earlier than A*. Clearly, the runtime is heavily dependent on the states we choose to expand and hence on the priority function $h_{\text{FOCAL}}$ used to sort FOCAL. Different instances of BSS and BCS use different priority functions. For example, $w$A* is a BSS that uses $h_{\text{FOCAL}}(n) = g(n) + wh(n)$, and Potential Search (PS) is a BCS that uses $h_{\text{FOCAL}}(n) = (C - g(n))/h(n)$ (henceforth referred to as the *potential function*).

It has already been shown that $h_{\text{FOCAL}}$ can be used in both definitions of FOCAL in the context of BSS and BCS [Gilon *et al.*, 2016]. This is also the case in the context of our paper, that is, $h_{\text{FOCAL}}$ can be used in both definitions of FOCAL in the context of anytime BSS and anytime BCS. However, not all priority functions are alike – some enable a more efficient search in any given iteration or more efficient reuse of previous search efforts. Thus, we identify the following two properties of priority functions.

**Definition 2** ($w$-admissible $h_{\text{FOCAL}}$)**.** A priority function $h_{\text{FOCAL}}(n)$ is *$w$-admissible* iff, for every expanded state $n$, $g(n) \leq wg^*(n)$, where $g^*(n)$ is the distance from the start state to $n$.

$w$-admissible $h_{\text{FOCAL}}$, such as in case of $w$A* [Likhachev, 2005], enable more efficient search because the bounded-suboptimality is guaranteed even when every state is expanded at most once. Unfortunately, this is not the case for any $h_{\text{FOCAL}}$, as exemplified by the graph in Figure 2. $S$ denotes the start state and $G$ denotes the goal state. Assume that $w = 2$ and $h_{\text{FOCAL}}$ is reverse alphabetical order. After expanding $S$, both $A$ and $C$ are in OPEN with $g(A) = 1$, $f(A) = 11$, $g(C) = 8$, and $f(C) = 16$. Since $f_{min} = 11$, both $A$ and $C$ are in FOCAL. $C$ is expanded next with $g(C) = 8$ as $h_{\text{FOCAL}}$ is reverse alphabetical order. Observe that $g^*(C) = 3$ and thus we expand $C$ with $g(C) > wg^*(C)$.

Avoiding state re-expansions in FS can violate the bounded suboptimality guarantee. This is exemplified by the graph in Figure 3. $S$ denotes the start state and $G$ denotes the goal state. Assume that $w = 2$ and $h_{\text{FOCAL}}$ is reverse alphabetical order. For simplicity, we assume that all heuristic values are zero. When expanding every state at most once, FS has the following trace: ('$\emptyset$' represents 'not in FOCAL'.)

| | OPEN | $f(n)(= g(n))$ | $h_{\text{FOCAL}}$ |
|---|---|---|---|
| Expand $S$ | $A$ | 1 | 2 |
| ($wf_{min} = 2$) | $B$ | 2 | 1 |
| Expand $B$ | $A$ | 1 | 1 |
| ($wf_{min} = 2$) | $C$ | $2 + \epsilon$ | $\emptyset$ |
| | $G$ | 4 | $\emptyset$ |
| Expand $A$ | $C$ | $2 + \epsilon$ | 2 |
| ($wf_{min} = 4 + 2\epsilon$) | $G$ | 4 | 1 |
| Expand $G$ | $C$ | $2 + \epsilon$ | 1 |

Thus, FS terminates after expanding $G$ with $g(G) = 4$, while the optimal solution's cost is $1 + 3\epsilon$ (for $\epsilon < 1$). We can easily choose $\epsilon$ such that the returned solution's cost is not within the suboptimality bound. In fact, we can extend this example with additional gadgets (shown in Figure 3 in grey) to make the solution's suboptimality arbitrarily bad.

The $w$-admissible property affects the efficiency of any one iteration of FS. In the next section, we discuss the anytime setting which involves consecutive iterations of FS. The following property affects the efficiency of reusing search efforts between such consecutive iterations.

**Definition 3** (Efficiently reusable $h_{\text{FOCAL}}$)**.** A priority function $h_{\text{FOCAL}}(n)$ is *efficiently reusable* iff it does not depend on $w$ or $C$.

The priority functions of $w$A* and PS are not efficiently reusable. Thus, any change in $w$ or $C$ may require reordering FOCAL, which is a costly operation. As we discuss in the next section, anytime algorithms repeatedly tighten their bounds. Thus, the efficiently reusable property can bear significant implications on their runtimes. For example, although ARA* is efficient due to the $w$-admissible property, its $h_{\text{FOCAL}}$ is not efficiently reusable and thus ARA* may still have to reorder its FOCAL between search iterations. Another example is the potential function, used in ATPS/ANA*, which is neither $w$-admissible nor efficiently reusable.

Another state-of-the-art BSS algorithm that is closely related to FS is explicit estimation search (EES) [Thayer and Ruml, 2011]. EES maintains three lists: The first list is $\text{OPEN}_f$, which is equivalent to OPEN as defined previously. The second list is $\text{OPEN}_{\hat{f}}$, which includes all states in $\text{OPEN}_f$ but is sorted according to $\hat{f}(n) = g(n) + \hat{h}(n)$, where $\hat{h}(n)$ is a (possibly inadmissible) estimate of the cost-to-goal. The third list is $\text{FOCAL}_{\hat{d}}$, which includes all the nodes in $\text{OPEN}_{\hat{f}}$ with $\hat{f}(n) \leq w\hat{f}(n)$ and is sorted according to $\hat{d}(n)$, a (possibly inadmissible) estimate of the edges-to-goal. Unlike FOCAL, one cannot simply expand states from $\text{FOCAL}_{\hat{d}}$ while maintaining suboptimality guarantees because $\hat{h}$ may be inadmissible. Thus, EES uses the following rule when expanding a state: If $f(\text{head}(\text{FOCAL}_{\hat{d}})) \leq wf_{min}$, then expand $\text{head}(\text{FOCAL}_{\hat{d}})$. Otherwise, if $f(\text{head}(\text{OPEN}_{\hat{f}})) \leq$

$wf_{min}$, then expand $\texttt{head}(\texttt{OPEN}_{\hat{f}})$. Otherwise, expand $\texttt{head}(\texttt{OPEN}_f)$. Thus, EES does not fit our formulation of FS although it terminologically uses a focal list.

### 2.3 Pseudocode

**Algorithm 1**: Focal Search (FS).
$n_{start}$ is the start state; $\texttt{isGoal}(n)$ is a predicate that returns true iff $n$ is a goal state; $\texttt{succ}(n)$ returns a list of all successors of $n$; and $w$ (or $C$) is the suboptimality bound (or cost bound). Blue (red) represents pseudocode relevant for BSS (BCS) only.

**Input:** $n_{start}$, $\texttt{isGoal}(n)$, $\texttt{succ}(n)$, $w$ (or $C$).
**Output:** A solution.

```
1  OPEN = FOCAL = {n_start}
2  return findPath(w (or C))
3  Procedure findPath(w (or C)):
4      while FOCAL ≠ ∅ do
5          f_min ← f(head(OPEN))
6          n ← head(FOCAL)
7          FOCAL ← FOCAL \ {n}
8          OPEN ← OPEN \ {n}
9          if isGoal(n) then
10             return solution
11         for each n' ∈ succ(n) do
12             OPEN ← OPEN ∪ {n'}
13             if f(n') ≤ wf_min (or C) then
14                 FOCAL ← FOCAL ∪ {n'}
15         if OPEN ≠ ∅ and f_min < f(head(OPEN)) then
16             updateLowerBound(wf_min, wf(head(OPEN)))
17     return "no solution exists"
18 Procedure updateLowerBound(old_b, new_b):
19     for each n ∈ OPEN do
20         if (f(n) > old_b) ∧ (f(n) ≤ new_b) then
21             FOCAL ← FOCAL ∪ {n}
```

Algorithm 1 presents pseudocode for the unified view of BSS and BCS (blue for BSS and red for BCS). Procedure findPath (line 3) implements FS, which starts with a singleton OPEN and FOCAL containing the start state $n_{start}$. The main loop of FS is conditioned on a non-empty FOCAL. Inside this loop, we first pop the head of FOCAL and remove it from OPEN as well (lines 5-8). If the popped state is a goal state, we return the solution found and terminate (lines 9-10). Otherwise, we generate its successors and add them to OPEN and possibly FOCAL (lines 11-14) (only if their $g$-values improve, that is, $g(n') \leq g(n) + c(n, n')$, where $c(n, n')$ is the transition cost from state $n$ to its successor state $n'$). In case the $f$-value of the head of OPEN increases as a consequence of the above operations, we need to update FOCAL accordingly (lines 15-16). Finally, if FOCAL is empty, we report that no solution exists (line 17).

## 3 Anytime Focal Search

Anytime search algorithms find a solution quickly and continue the search process to find improved solutions until time runs out. They are useful for solving combinatorial problems when deliberation time is limited. In this section, we present AFS. Like ARA* and ATPS/ANA*, AFS finds an optimal solution given enough time, provides suboptimality guarantees for each search iteration, and reuses previous search efforts. However, AFS can compute tighter suboptimality bounds than ATPS/ANA* and, unlike ARA* and ATPS/ANA*, it can use an arbitrary $h_{\texttt{FOCAL}}$. Moreover, AFS may reuse previous search efforts more efficiently than ARA* and ATPS/ANA* if $h_{\texttt{FOCAL}}$ is efficiently reusable.

We start by discussing different ways of changing the (suboptimality or cost) bound between consecutive search iterations.

### 3.1 Anytime Bounds

Denote the costs of the solutions found in progressive search iterations of an anytime algorithm by $S_1, S_2, \ldots$.

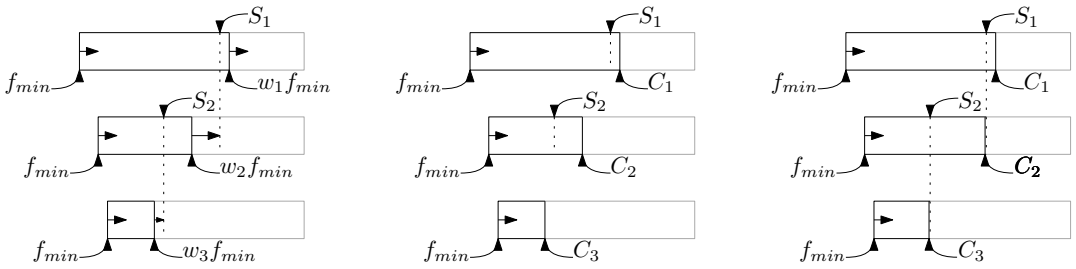
**Definition 4** (Bounds update scheme)**.** Three ways to update the (suboptimality or cost) bound between consecutive search iterations are as follows:

1. Given a sequence $w_1 > \ldots > w_K = 1$, search iteration $i$ uses $w_i$ as the suboptimality bound.
2. Given a sequence $C_1 = \infty > \ldots > C_K$, search iteration $i$ uses $C_i$ as the cost bound.
3. The first search iteration uses cost bound $C_1 = \infty$. In search iteration $i > 1$, we adaptively update the cost bound based on $S_{i-1}$. One common choice is $C_i = S_{i-1} - \epsilon$ (which is equivalent to suboptimality bound $w_i = \frac{S_{i-1}}{f_{min}} - \epsilon$), where $\epsilon$ is a small positive number and $f_{min}$ is the $f$-value of the head of OPEN when search iteration $i-1$ terminates.

In the anytime BSS framework, Definition 4(1) is commonly used. Using $w_1 > \ldots > w_K$ guarantees a sequence of solutions, each with a better suboptimality guarantee than the previous ones. However, just using $w_1 > \ldots > w_K$ does not guarantee that the sequence of solutions has strictly decreasing costs, that is, it is not necessarily the case that $S_i > S_{i+1}$. From a user's perspective, it seems reasonable to expect that an anytime algorithm produces solutions with strictly decreasing costs as time progresses, and, ideally, with "diminishing returns," that is, the algorithm converges quickly to a "good" solution.

One way to accommodate this expectation is to use $S_{i-1}$ as a cost bound for search iteration $i$. We can use this cost bound to prune *surplus states*, that is, in search iteration $i$, when generating a state with a cost higher than the current cost bound $S_{i-1}$ we do not add it to OPEN or FOCAL. Furthermore, when popping such a state from the head of FOCAL (lines 6-7 in Algorithm 1), we do not "process" it (lines 9-14 in Algorithm 1) and instead continue immediately to the next state in FOCAL. With these modifications, an anytime BSS algorithm guarantees that the sequence of solution costs are strictly decreasing, that is, $S_i > S_{i+1}$ for every $i$. Since $w_K = 1$, it is guaranteed to eventually find an optimal solution. Figure 4(a) illustrates the iterative behavior of FOCAL in this update scheme.

Definition 4(2) fits the anytime BCS framework. Unlike the anytime BSS framework, here, there is no guarantee to eventually find an optimal solution for an arbitrary sequence of




(a) According to Definition 4(1) (b) According to Definition 4(2) (c) According to Definition 4(3)

Figure 4: Illustrates the anytime effect on FOCAL for the three different ways of updating the bounds.

cost bounds $C_1 > \ldots > C_K$. Figure 4(b) illustrates the iterative behavior of FOCAL in this update scheme.

Definition 4(3) is commonly used in the anytime BCS framework [Stern *et al.*, 2014] when we want to guarantee that we eventually find an optimal solution. Here, too, we start with $C_1 = \infty$ but in later search iterations update the cost bound adaptively with respect to the cost of the best solution found so far. Figure 4(c) illustrates the iterative behavior of FOCAL in this update scheme. This update scheme essentially unifies the previous two update schemes if we prune surplus states. In fact, ATPS/ANA* uses this update scheme along with the potential function for prioritization of states in FOCAL. Thus, ATPS/ANA* is a special case of AFS. When using Definition 4(3), using $S_{i-1}$ is better than $w_i f_{min}$ since it is at least as tight a bound but perhaps tighter (that is, if $f_{min}$ increases during search iteration $i$). The pruning of surplus nodes then effectively implements the cost bound $S_{i-1}$, and – like in ATPS/ANA* – it thus suffices to maintain only FOCAL since OPEN and FOCAL are identical.

## 3.2 Pseudocode

**Algorithm 2**: Anytime Focal Search (AFS).
$n_{start}$ is the start state; and getNextBound() is a specification of one of the update schemes in Definition 4. Blue (red) represents pseudocode relevant for BSS (BCS) only.

**Input:** $n_{start}$, getNextBound().
**Output:** Solution(s)).

```
1  OPEN = FOCAL = {n_start}
2  while search not halted or optimal solution not found do
3      w (or C) ← getNextBound()
4      updateFocalBound(w f(head(OPEN)) (or C))
5      if h_FOCAL is not efficiently reusable then
6          reorder FOCAL
7      sol ← findPath(w (or C))
8      if sol = no-solution then
9          break
10     report sol
11 Procedure updateFocalBound(new_b):
12     for each n'' ∈ FOCAL do
13         if f(n'') > new_b then
14             FOCAL ← FOCAL \ {n''}
```

Algorithm 2 presents the pseudocode for AFS. AFS uses a specification of one of the update schemes in Definition 4. The main loop of AFS (line 2) is conditioned on the availability of runtime and the suboptimality of the best solution found so far[4]. Inside this loop, AFS calls FS with the current (suboptimality or cost) bound as an argument (line 7). After each search iteration terminates, FOCAL is updated to ensure that all of its states are within the new (suboptimality or cost) bound (line 4). If $h_{\text{FOCAL}}$ is not efficiently reusable, FOCAL is reordered (line 6).

## 3.3 Theoretical Properties

FS and AFS are different from ARA* in that they may require state re-expansions within the same search iteration to guarantee finding solutions with costs within the (suboptimality or cost) bound if $h_{\text{FOCAL}}$ is not $w$-admissible. While state re-expansions may result in longer runtimes of some search iterations, they allow AFS to provide suboptimality guarantees for each search iteration. Such (suboptimality or cost) bounds are important since it is not known in advance when an anytime algorithm is forced to terminate. Moreover, AFS does not need to maintain any additional lists, such as INCONS in ARA*. This makes AFS simpler to understand and implement.

**Theorem 1.** *In each search iteration $i$, if AFS reports a solution with cost $S_i$, it is guaranteed that $S_i \leq w_i S^*$, where $S^*$ is the cost of an optimal solution.*

*Proof.* Follows directly from the bounded suboptimality guarantees of FS. □

We now prove that AFS with the potential function computes tighter suboptimality bounds than ATPS/ANA*. This is because FS maintains $f_{min}$ at all times and its suboptimality bound is $S/f_{min}$ while PS has a suboptimality bound of $\max_{n \in \text{FOCAL}}(C - g(n))/h(n)$ [Stern *et al.*, 2014]. Here, $S$ is the cost of the solution found by both FS and PS, $f_{min}$ is the $f$-value of the head of OPEN when FS terminates and $C$ is the cost bound used by PS.

**Lemma 2.** *Let $B_{PS}$ and $B_{FS}$ be the suboptimality bound computed by PS and FS, respectively. $B_{FS} \leq B_{PS}$.*

*Proof.*

$$B_{PS} = \max_{n \in \text{FOCAL}} \frac{C - g(n)}{h(n)}.$$

[4]When $f_{min}$ equals the cost of the best solution found so far we can terminate with the optimal solution.

Since $\frac{C-g(n)}{h(n)} \geq 1$ and $g(n) \geq 0$ for every $n$ in FOCAL,

$$B_{PS} \geq \max_{n \in \text{FOCAL}} \frac{C - g(n) + g(n)}{h(n) + g(n)} = \max_{n \in \text{FOCAL}} \frac{C}{f(n)}.$$

Since $C$ is a constant and $C \geq S$,

$$B_{PS} \geq \frac{C}{\min_{n \in \text{FOCAL}} f(n)} = \frac{C}{f_{min}} \geq \frac{S}{f_{min}} = B_{FS}.$$

Hence, $B_{FS} \leq B_{PS}$. □

The fact that AFS computes tighter suboptimality bounds than ATPS/ANA* can have implications on the anytime behavior because it allows AFS to decrease the bound faster. AFS is also more general than ATPS/ANA* because ATPS/ANA* is a special case of AFS in which FOCAL is sorted according to a specific priority function (namely, the potential function), while AFS allows for arbitrary priorities. Unlike ATPS/ANA*, when the priorities for ordering FOCAL are efficiently reusable, AFS is not required to iterate over FOCAL and reorder it when the bound changes between search iterations. This could translate to substantial time savings when FOCAL is large or when solutions are found frequently. Moreover, AFS also facilitates anytime search in domains, as in one of our experimental domains, that have no well-defined heuristic function, and hence no useful definition of potential function. Here, AFS is still viable but ATPS/ANA* is not. Finally, the flexibility with arbitrary priorities in AFS allows incorporating domain-specific knowledge. This, in turn, can guide the search process better.

## 4 Experimental Results

We now demonstrate the usefulness of AFS for solving hard combinatorial problems. We choose two NP-hard problems for our experiments: 1) the generalized covering traveling salesman problem (GCTSP) and 2) the multi-agent path finding (MAPF) problem. For GCTSP, we show the runtime advantage of AFS over other anytime algorithms, which stems from its ability to use domain-specific priority functions. For MAPF, we show the broader applicability of AFS compared to other anytime algorithms. Here, a domain-specific $h_{\text{FOCAL}}$ is informative while, in fact, no non-zero admissible $h$ is currently known. This makes the other anytime algorithms discussed in this paper inapplicable to this domain. In both domains, we use Definition 4(3) as the bound update scheme.

### The Generalized Covering Traveling Salesman Problem

The GCTSP [Shaelaie *et al.*, 2014] is defined by an undirected graph that has one depot vertex and other vertices called facilities. Weighted edges between vertices represent distances. Each facility has a set of costumers associated with it, and customer $i$ has a prize $p_i$. A customer can be covered by more than one facility. The task in GCTSP is to find a tour that starts at the depot and collects a specified minimum prize $P$ while minimizing the total distance traveled. A tour collects the prizes of all customers associated with any of its facilities. In GCTSP's search space, each state represents a partial tour of facilities with its cumulative prize [Pohl, 1973; Thayer and Ruml, 2008]. The successors of a state $n$ augment

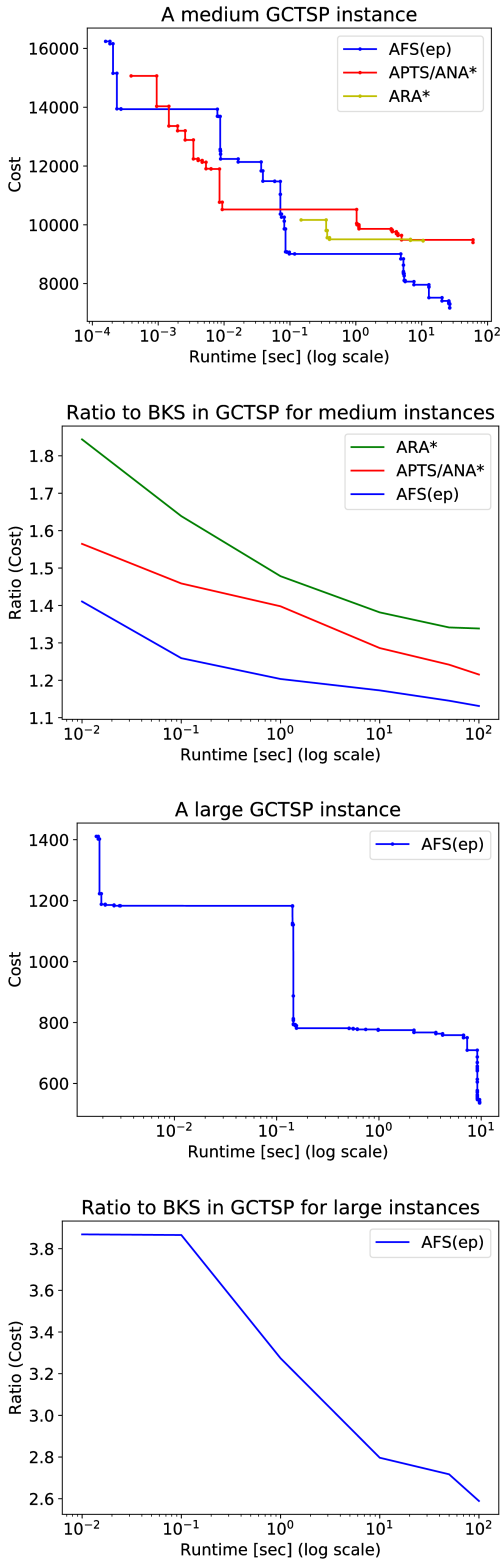



Figure 5: Shows the behaviors of anytime BSS and BCS algorithms in the GCTSP domain. The first and third panels show behaviors on typical medium and large size instances, respectively. The second and fourth panels show aggregate behaviors on 69 medium and 14 large instances, respectively. BKS stands for best known solution.

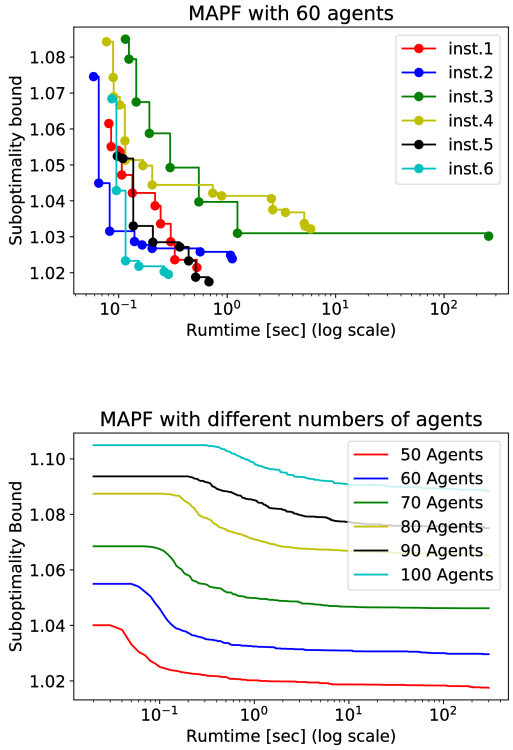



Figure 6: Shows the behavior of AFS in the MAPF domain. The first panel shows AFS's typical behavior on a few random instances with 60 agents each. The second panel shows AFS's aggregate behavior on different numbers of agents with 50 instances each. All experiments use a $32 \times 32$ four-neighbor grid with 20% blocked cells placed randomly.

a non-visited facility to $s$. We define the heuristic value of a state $n$ with cumulative prize $cp$ to be $h(n) = H(P - cp)$. $H(P - cp)$ is the minimum distance from the depot to any state with prize $P - cp$. This heuristic is admissible and is pre-computed using Dijkstra's algorithm.

We evaluate AFS in the GCTSP domain on benchmark instances from [Shaelaie *et al.*, 2014]. We use 69 medium instances (between 100 and 200 vertices) and 14 large instances (between 535 and 1000 vertices). AFS uses the cumulative prize multiplied by the potential function as its priority function. We also evaluate ATPS/ANA* and ARA*. All runs have a time limit of 100 seconds. Figure 5 shows the results. On medium-sized instances, AFS convincingly beats ATPS/ANA* and ARA*. More significantly, both ATPS/ANA* and ARA* fail to find any solution within the time limit for any of the 14 large sized instances. These results suggest that adding domain-dependent knowledge to the priority function, as allowed in the general framework of AFS, has significant runtime benefits.

### The Multi-Agent Pathfinding Problem

Given an undirected graph and a set of agents with unique start and goal vertices, the MAPF problem is to find collision-free paths for all agents from their start vertices to their goal vertices. The agents traverse edges in unit time but can also wait at vertices. Here, we consider minimizing the solution cost given by the sum of travel times of agents along their paths, which is known to be NP-hard [Yu and LaValle, 2013]. Conflict-Based Search (CBS) [Sharon *et al.*, 2015] is a state-of-the-art MAPF solver. CBS uses a two-level search. On the high level, a search is performed on a constraint tree. In this constraint tree, each state represents a set of constraints imposed on the motions of individual agents. On the low level, single-agent searches are performed such that none of the constraints imposed by the relevant high-level states are violated. We adapt BCBS [Barer *et al.*, 2014] to AFS. BCBS$(w, 1)$ is a variant of CBS that uses focal search with suboptimality bound $w$ to conduct the high-level search and A* to conduct the low-level search.

The high-level search of CBS uses the paths lengths of the agents in a high-level state as its $g$-value. It does not have non-zero admissible $h$-values. Thus, we cannot apply ATPS/ANA* in this domain because the potential function is undefined, and we cannot apply ARA* in this domain because $f(n) = g(n) + wh(n) = g(n)$. On the other hand, the number of collisions between paths of agents in a high-level state is an informative but inadmissible estimate for the cost-to-goal. Fortunately, the general framework of AFS allows us to use this informative estimate in $h_{\texttt{FOCAL}}$ and, moreover, it is efficiently reusable. In fact, to the best of our knowledge, this adaptation of AFS constitutes the first anytime MAPF solver. Hence, the experimental results in Figure 6 report only on the performance of AFS. We observe that this adaptation of AFS exhibits the diminishing returns property that is characteristic of good anytime behavior.

## 5 Conclusions

In this paper, we presented AFS, an anytime version of FS that unifies the anytime variants of BSS and BCS. We also emphasized the generality of AFS and showed how other state-of-the-art anytime search algorithms, like ARA* and ATPS/ANA*, are special cases of it. Theoretically, we proved the correctness and bounded suboptimality of AFS, the better quality of its bounds compared to ATPS/ANA*, and its ability to efficiently reuse previous search efforts when it does not need to reorder FOCAL between search iterations. Empirically, we demonstrated the benefits of incorporating domain-specific knowledge in $h_{\texttt{FOCAL}}$.

Finally, the success of AFS in the GCTSP and MAPF domains is illustrative of a more general advantage of its framework. When admissible estimates of the costs-to-goal are available, AFS can always use them in $h$. When the available estimates are inadmissible but informative, AFS gives us the important option to use them in $h_{\texttt{FOCAL}}$. Indeed, for many hard combinatorial problems, efficient approximation algorithms produce such inadmissible but informative estimates of cost-to-goal. Moreover, abstractions and relaxations of search problems are admissible but not always informative. While their additive combinations may not be admissible, they are often informative and can be used in the AFS framework while providing bounded-suboptimality guarantees.

## References


[Aine and Likhachev, 2016] Sandip Aine and Maxim Likhachev. Search portfolio with sharing. In *Proceedings of the 26th International Conference on Automated Planning and Scheduling*, 2016.

[Barer *et al.*, 2014] Max Barer, Guni Sharon, Roni Stern, and Ariel Felner. Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem. In *Proceedings of the 7th Annual Symposium on Combinatorial Search*, 2014.

[Bonet and Geffner, 2001] Blai Bonet and Héctor Geffner. Planning as heuristic search. *Artificial Intelligence*, 129:5–33, 2001.

[Gilon *et al.*, 2016] Daniel Gilon, Ariel Felner, and Roni Stern. Dynamic potential search - A new bounded suboptimal search. In *Proceedings of the 9th Annual Symposium on Combinatorial Search*, 2016.

[Hart *et al.*, 1968] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems, Science, and Cybernetics*, SSC-4(2):100–107, 1968.

[Hatem and Ruml, 2014] Matthew Hatem and Wheeler Ruml. Simpler bounded suboptimal search. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence*, 2014.

[Helmert and Roger, 2008] Malte Helmert and Gabriele Roger. How good is almost perfect. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence*, 2008.

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

[Likhachev *et al.*, 2003] Maxim Likhachev, Gordon Geoffrey, and Sebastian Thrun. ARA* : Anytime A* with provable bounds on sub-optimality. In *Advances in Neural Information Processing Systems 16*, 2003.

[Likhachev, 2005] Maxim Likhachev. *Search-based Planning for Large Dynamic Environments*. PhD thesis, 2005.

[Pearl and Kim, 1982] Judea Pearl and Jin Kim. Studies in semi-admissible heuristics. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 4:392 –399, 1982.

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

[Pohl, 1973] Ira Pohl. The avoidance of (relative) catastrophe, heuristic competence, genuine dynamic weighting and computational issues in heuristic problem solving. In *Proceedings of the 3rd International Joint Conference on Artificial Intelligence*, 1973.

[Shaelaie *et al.*, 2014] Mohammad H. Shaelaie, Majid Salari, and Zahra Naji-Azimi. The generalized covering traveling salesman problem. *Applied Soft Computing*, 24:867 – 878, 2014.

[Sharon *et al.*, 2015] Guni Sharon, Roni Stern, Ariel Felner, and Nathan R. Sturtevant. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*, 219:40–66, 2015.

[Stern *et al.*, 2011] Roni Stern, Rami Puzis, and Ariel Felner. Potential search: A bounded-cost search algorithm. In *Proceedings of the 21st International Conference on Automated Planning and Scheduling*, 2011.

[Stern *et al.*, 2014] Roni Stern, Ariel Felner, Jur van den Berg, Rami Puzis, Rajat Shah, and Ken Goldberg. Potential-based bounded-cost search and anytime non-parametric A*. *Artificial Intelligence Journal*, 214:1–25, 2014.

[Thayer and Ruml, 2008] Jordan Thayer and Wheeler Ruml. Faster than weighted A*: An optimistic approach to bounded suboptimal search. In *Proceedings of the 18th International Conference on Automated Planning and Scheduling*, 2008.

[Thayer and Ruml, 2010] Jordan Thayer and Wheeler Ruml. Anytime heuristic search: Frameworks and algorithms. In *Proceedings of the 2nd Annual Symposium on Combinatorial Search*, 2010.

[Thayer and Ruml, 2011] Jordan Thayer and Wheeler Ruml. Bounded suboptimal search: A direct approach using inadmissible estimates. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, 2011.

[van den Berg *et al.*, 2011] Jur van den Berg, Rajat Shah, Arthur Huang, and Kenneth Y. Goldberg. Anytime nonparametric A*. In *Proceedings of the 25th Conference on Artificial Intelligence*, 2011.

[Wilt and Ruml, 2012] Christopher Wilt and Wheeler Ruml. When does weighted A* fail? In *Proceedings of the 5th Annual Symposium on Combinatorial Search*, 2012.

[Wilt and Ruml, 2016] Christopher Wilt and Wheeler Ruml. Effective heuristics for suboptimal best-first search. *Journal of Artificial Intelligence Research*, 57:273–306, 2016.

[Yu and LaValle, 2013] Jingjin Yu and Steven M. LaValle. Structure and intractability of optimal multi-robot path planning on graphs. In *Proceedings of the 27th AAAI Conference on Artificial Intelligence*, 2013.