Revisiting Suboptimal Search

Jingwei Chen and **Nathan R. Sturtevant**
University of Alberta
Edmonton, Alberta, Canada
{jingwei5, nathanst}@ualberta.ca

William Doyle and **Wheeler Ruml**
University of New Hampshire
Durham, New Hampshire, USA
{doyle, ruml}@cs.unh.edu

Abstract

Suboptimal search algorithms can often solve much larger problems than optimal search algorithms, and thus have broad practical use. In the past decade several algorithms have been proposed that improve performance over the basic weighted A*, but these algorithms rely special heuristics and data structure implementations, meaning that they require significantly more effort to implement and use than basic weighted A*. The goal of this paper is to provide a simple algorithm that is easy to implement and can achieve state-of-the-art performance. The paper does this in three ways. First, it studies the influence of node re-openings during search, showing the potential savings and cost of re-openings. As most existing suboptimal algorithms re-open states, turning off re-openings can often improve performance. Second, it studies termination conditions, providing a better termination condition for approaches like Optimistic search. Finally, taken together with recently-developed priority functions, a general framework for Improved Optimistic Search is developed that is both simpler than existing algorithms and also has better performance in

1 Introduction

Most real-world problems are too large or have significant other constraints that prevent finding optimal solutions. This motivates a broad literature in suboptimal search, as allowing even a small amount of suboptimality can significantly decrease the time needed to find a solution. The classic algorithm for suboptimal search is Weighted A* (WA*) (Pohl 1970), which finds solutions that have cost which is at most w times the optimal solution cost. There are many variants of suboptimal solvers, including those with unbounded solution length (such as Greedy Best First Search (Russell and Norvig 2009)), and those with bounds on the solution length produced (Stern, Puzis, and Felner 2011; Valenzano et al. 2013). Research has also addressed the use of multiple heuristics (Thayer and Ruml 2011) or customized heuristics (Wilt and Ruml 2015) that can improve the performance of suboptimal solvers.

This paper studies the simple setting of w-bounded suboptimal search with a single heuristic function, by building on algorithms like WA*, A$^*_\epsilon$ (Pearl and Kim 1982), and Optimistic Search (Thayer and Ruml 2008). In addition to building a more general parameterized algorithm, Improved Optimistic Search (IOS), the paper also develops improved termination conditions and performs a general study of the problem of node re-expansions in WA*.

Like Optimistic Search, IOS uses a focal list to search and find a solution, and an open list to prove the suboptimality of the solution found. But, in many cases the maximum f-cost of a state expanded in the focal search can be used to prove that the solution is within the optimality bound without expanding states in the open list. IOS is also able to use a broader class of priority functions (Chen and Sturtevant 2019) that have recently been developed and are compatible with the IOS framework.

Summarizing this, IOS represents a new family of algorithms for which many re-expansions can be avoided, that can use new classes of priority functions, and that has better termination conditions. IOS is not only much simpler that most existing algorithms, but also has similar or improved performance on a broad class of empirical domains.

2 Background and Related Work

Shortly after the development of heuristic search algorithms (Hart, Nilsson, and Raphael 1968), suboptimal variants were also developed, beginning with WA*. WA* uses a priority function of $f(n) = g(n) + w \cdot h(n)$ and returns w-optimal solutions, or solutions that are no more than w times longer than the optimal solution. WA* has widespread use because it has good performance and is very simple to implement, using the same data structures as A* – typically a binary heap. One particularly useful feature of WA* is that it can maintain its optimality bound even if it does not re-open closed states when it finds a shorter path to them during search. It can, however, re-open closed states if desired (Sepetnitsky, Felner, and Stern 2016).

Looking more broadly, suboptimal search algorithms have two tasks. The first is finding a solution, and the second is proving that the solution is within the suboptimality bound. While WA* handles these tasks together, it can be more efficient to separate these concerns. The most common way of separating these concerns is through dividing states into both an open list and a focal list (Pearl and Kim 1982). The open list is sorted by $g(n)+h(n)$ and is used for proving the suboptimality bound, while the focal list is sorted by a

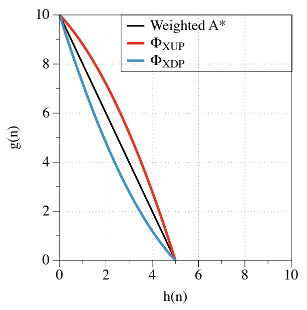

Figure 1: Sample isolines associated with WA*, Φ_{XDP} and Φ_{XUP}.

different metric, and is used for finding a solution. The open list is required to re-expand states that have been previously expanded in focal in order to prove optimality.

A$^*_\epsilon$ (Pearl and Kim 1982) was the first focal list algorithm, where the focal list was a of states subset taken from the open list and sorted with different priority. The re-expansion policy for A$^*_\epsilon$ is not explicitly described, but the typical assumption is that it performs re-expansions whenever shorter paths are found to a previously expanded state (whether from focal or open). A$^*_\epsilon$ only expands states from focal that have already been proven to be within the suboptimality bound.

Optimistic search (Thayer and Ruml 2008) is another focal list algorithm that performs re-expansions during search. The primary change between Optimistic search and A$^*_\epsilon$ is that Optimistic search uses the focal list to find a solution before expanding the open list and trying to prove that the solution is within the suboptimality bound. Like A$^*_\epsilon$, Optimistic search performs re-openings both on focal and open when shorter paths are found. When trying to find solutions that are w-optimal, the Optimistic search paper suggest exploring the focal list using WA* with weight of $w_f = 2w-1$ when a suboptimality bound of w is desired, although no theoretical justification is provided for this weight. Optimistic search can use similar data structures as A*, it just requires two priority queues.

Explicit Estimate Search (EES) (Thayer and Ruml 2011) and its variants use additional heuristics, learned estimates, and priority queues to estimate the search effort required to reach the goal in addition to the distance to the goal. While this can make EES efficient in practice, EES is more difficult to implement than other algorithms as EES requires three priority queues, a heuristic function, and a distance function. At least one of these queues is typically implemented using balanced binary trees for efficiency purposes.

Dynamic Potential Search (DPS) (Gilon, Felner, and Stern 2016) uses a dynamic priority function which can be implemented using a single priority queue. But, because of the dynamic nature of the search, DPS must occasionally re-order the states on the priority queue. This can be done more efficiently if costs can be partitioned into buckets, but like other focal list algorithms DPS must also re-open closed states when it finds a shorter path.

Recent work (Chen and Sturtevant 2019) has studied the issue of node re-expansions and suggested alternate priority functions that can be used with best-first search. In particular, a class of function $\Phi(x, y)$ are proposed for which best-first search can use a priority function of $f(n) = \Phi(h(n), g(n))$. Given certain restrictions on Φ, it can be shown that closed states do not need to be re-opened during search to maintain a w-optimal solution bound. Two particular priority functions are of interest, the convex downward curve $\Phi_{XDP}(x, y) = \frac{1}{2w}[y + (2w-1)x + \sqrt{(y-x)^2 + 4wyx}]$ and the convex upward curve $\Phi_{XUP}(x, y) = \frac{1}{2w}(y + x + \sqrt{(y+x)^2 + 4w(w-1)x^2})$. Sample isolines for all states with a given priority are shown in Figure 1. The slope of these curves shift where the suboptimality in the path is allowed. The convex downward curve (Φ_{XDP}) requires that the path is near-optimal near the start, with greater suboptimality allowed near the goal, while the convex upward curve (Φ_{XUP}) requires the path to be near-optimal near the goal. This contrasts with WA* which permits uniform suboptimality across the entire search path.

3 Problem Definition

A suboptimal heuristic search problem is defined by an n-tuple $(s, g, succ(), c(), h(), w)$, where s is the start state, g is the goal state, $succ$ is a successor function that computes the successors of a given state, c is a cost function that provides the cost to move from a state to its successor, and h is a heuristic function that estimates the cost between any state and the goal. Let $d(a, b)$ be the optimal cost between any two states. It is assumed that the heuristic is admissible $\forall_n h(n, g) \leq d(n, g)$ and consistent $h(a) \leq h(b) + d(a, b)$. Finally, w is the suboptimality bound on the solution returned. An algorithm that solves a suboptimal heuristic search problem must return a path between s and g that has cost at most w times the cost of the optimal solution. The g-cost of a node, $g(n)$, is the current path cost in a given priority queue.

4 Improved Optimistic Search

Improved Optimistic Search (IOS) is built on a general framework that uses two searches. One search is designed to optimistically find a solution as quickly as possible through expansions on FOCAL, while the other search is designed to prove that the solution found is within the bound through expansions on OPEN. Although portions of OPEN and FOCAL can be merged for efficiency purposes, these are logically treated as two separate searches. OPEN is a standard A* open list with $f(n) = g(n) + h(n)$. FOCAL uses a priority function $f_f(n)$ with a recommended bound of $w_f = 2w-1$. The exact priority function used in FOCAL is discussed in Section 4.5.

The general IOS algorithms is shown in Algorithm 1. If we ignore the re-opening policies (lines 13-18), the approach is very simple. It will expand the best state on FOCAL until an incumbent solution I is found (line 6), where $c(I)$ is the cost of the solution I. Then it will expand the best state on OPEN until w-optimality is proven (line 12). If re-openings are allowed, states re-opened on FOCAL

Algorithm 1 Improved Optimistic Search

```
1: procedure IMPROVED OPTIMISTIC SEARCH($start$, $goal$, $w$)
2:     Push($start$, OPEN)
3:     Push($start$, FOCAL)
4:     $I \leftarrow \varnothing$   [ $c(I) = \infty$) ]
5:     while $c(I)$ not $w$-optimal do
6:         if est. path length of best on FOCAL $< c(I)$ then
7:             Expand best from FOCAL
8:             if best == $goal$ then
9:                 $I \leftarrow path(best)$
10:            end if
11:        else
12:            Expand best from OPEN
13:            if child $s$ has shorter path to $s$ on FOCAL then
14:                // Choose one of the following policies:
15:                (a) Update cost of $s$ on FOCAL // Update
16:                (b) Re-open $s$ on FOCAL // Re-open
17:                if $s \in I$ then
18:                    (c) update cost of $I$ // Solution-update
19:                end if
20:            end if
21:        end if
22:    end while
23:    return $failure$
24: end procedure
```

in line 16 will then be re-expanded in line 6. Because some priority functions, such as Φ_{XUP}, are not directly estimating the length of the solution that will be found, line 6 explicitly uses the estimated path length.

Re-openings are one of a number of parameters that can be tuned in an implementation of IOS. Section 4.1 describes the termination conditions, Section 4.3 describes policies for re-opening states, Section 4.5 describes possible priority functions used in FOCAL, and Section 4.6 discusses the recommended bound of $w_f = 2w - 1$.

4.1 Termination and Proving Bounds

Existing algorithms, such as Optimistic search and EES, use the minimum f-cost of a state on OPEN, $f_{\min}$, to prove the optimality of the solution. In particular, $f_{\min}$ on OPEN is a lower-bound on the optimal solution cost. This holds because, with a consistent heuristic, the minimum f-cost in OPEN never decreases, and at the goal the f-cost is equivalent to the solution cost. Thus, a solution found in FOCAL with cost less than or equal to $w \cdot f_{\min}$ is guaranteed to be w-optimal. If the solution cannot immediately be proven to be optimal, states on OPEN are expanded until the bound on the solution quality is proven. While IOS uses this termination condition ($c(I) \leq wf_{\min}$), it can also use a second termination condition.

The typical priority function used with WA* is $f(n) = g(n) + wh(n)$. Under this priority function, the priority represents the estimated solution cost, because when the goal is reached, $h(n) = 0$ and $f(n) = g(n)$. An alternate formulation of this priority function, which will perform identically, is $f'(n) = \frac{g(n)}{w} + h(n)$. In this formulation, the f-cost is not an estimate of the solution cost that will be found, but an estimate of the optimal solution cost, akin to f_{min} in OPEN.

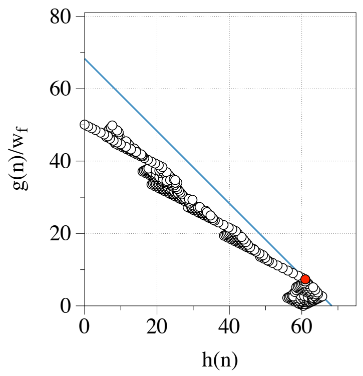

Figure 2: A plot of each state expanded in the focal list in a problem with $w_f = 2.0$ and $w = 1.5$.

Now, consider what happens in a WA* search with weight of $w_f > w$ using this alternate formulation of the priority function. Because WA* does not need to re-expand states to achieve w-optimality, when the search with weight w_f expands a state with priority $f'_{\max}$, this guarantees that the optimal solution has cost no less than $f'_{\max}$. Thus, any solution with cost less than or equal to $w \cdot f'_{\max}$ is guaranteed to be w-optimal. This implies that in IOS the maximum f-cost of any state expanded from FOCAL can be used to provide an additional termination condition for search. If WA* always found solutions that were exactly w-optimal, then this wouldn't be useful. But, in practice it typically finds solutions that are much closer to optimal than the bound. Thus, the second termination condition for IOS is to terminate when $c(I) \leq wf'_{\max}$, assuming $f'(n) = \frac{g(n)}{w} + h(n)$.

The effect of this bound is illustrated in Figure 2. This figure plots $(h(n), g(n)/w)$ for every state that is expanded in a WA* search problem with $w = 2$. Each point represents an estimate of the optimal solution cost. Because the solution was found with $f'(n) = 50.08$ (actual path cost 100.16), the optimal solution must be no shorter than 50.08. But, it is possible to get better bounds on the actual solution quality found. First, the start state has $f'(n) = 60.41$. Thus, the suboptimality is known to be no more than $100.16/60.41 = 1.66$. But, the state expanded with maximum f-cost (shown in red) has $f'(n) = 68.30$, so the suboptimality bound can be reduced to $100.16/68.30 = 1.47$.

Now, consider if IOS is searching with $w = 1.5$ and $w_f = 2$. On this problem, a WA* search using $w_f = 2$ will find and be able to prove that the solution found is w-optimal. No additional expansions in OPEN will be needed to prove the bound. Previous algorithms that are unaware of the improved bound would be required to expand the states until the minimum f in OPEN is raised to $100.16/1.5 = 66.77$.

This approach works in this problem because the maximum f-cost is found at a state other than the start state. This is related to the definition of the high-water mark used for analyzing greedy best-first search (Wilt and Ruml 2014; Heusner, Keller, and Helmert 2018). If the start state is in a local minima, then the maximum f-cost needed to escape

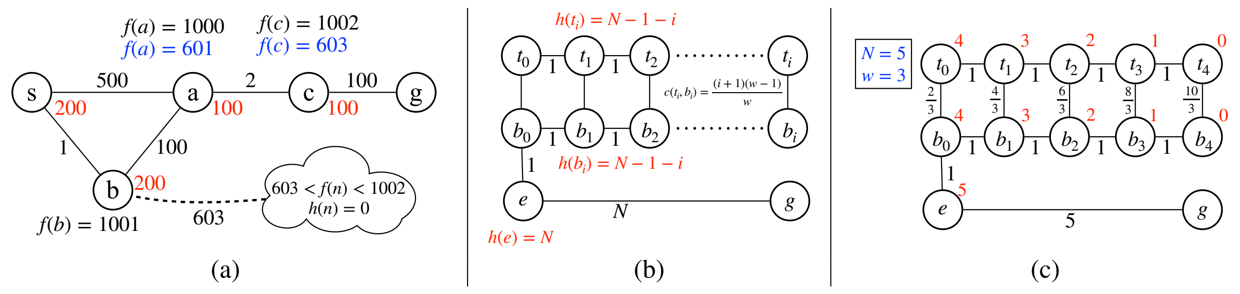

Figure 3: (a) Best-case from performing re-openings in an example with $w = 5$. (b) Generic example where re-openings require $O(N^2)$ re-expansions. (c) Specific instance of part (b) for $N = 5$ and $w = 3$.

the local minima will be larger than the f-cost of the start state and provide a better bound in practice.

4.2 Best and Worst-Case Analysis for Re-Expansions

Given the improved termination conditions, the next step is to study the impact of node re-openings and re-expansions during search. A node re-opening occurs when a shorter path to a node is found and the node is placed back on a priority queue. A re-expansion occurs when a node is expanded after being re-opened. This section demonstrates that node re-openings can have a significant negative or positive impact on performance. Previous work has studied the influence of re-openings on solution quality in WA* (Valenzano, Sturtevant, and Schaeffer 2014), but we are not aware of any work that shows worst-case bounds for the total number of expansions. These cases are similar to the analysis of inconsistent heuristics and BPMX (Felner et al. 2011), but the examples are unique to WA* and the underlying heuristics are consistent.

The first example, in Figure 3(a), shows how performing re-openings can provide arbitrarily large savings. In this figure edges are labeled with their cost in black. Nodes are labeled with their h-cost in red. If re-openings are not allowed, a WA* search with $w = 5$ will expand s, followed by a with $f(a) = 1000$ and b with $f(b) = 1001$. Then, because a cannot be re-opened, the cloud of nodes at the bottom, which can be arbitrarily large, will be expanded with $f(n) > 603$. However, if re-openings are allowed, then a will be re-opened with $f(a) = 601$, shown in blue. This leads to c being expanded with $f(c) = 603$, followed by the goal with $f(g) = 203$. With re-openings, only 6 expansion are required, but without re-openings, an arbitrarily large number are needed.

The second example, in Figure 3(b) shows the potential cost of performing re-openings. While this is an artificial example, the example is reflective of similar real-world problems that arise in grid pathfinding problems. Figure 3(b) is a scalable example that works for any weight $w > 1$ and any parameter for the problem size, N. The start state is t_0 and the goal is g. There is a top row of states t_i and a bottom row of states b_i with identical heuristics and unit edge costs between states in the same row. But, the cost of the edge between the top and bottom paths gradually increases along the path. After expanding the top row, the states going across the bottom row from left-to-right will have decreasing f-costs. Thus, the bottom nodes will be expanded right-to-left. After expanding each node, all subsequent nodes to the right in the row will be re-expanded, as the g-cost has decreased.

We provide a concrete instance of the general graph in Figure 3(c) for $N = 5$ and $w = 3$. Table 1 shows the order of the node expansions in this example. There are 22 total node expansions; b_4 is expanded $N = 5$ times, b_3 is expanded 4 times, and so on.

In general, the top nodes (t_i) will each be expanded once. The bottom nodes (b_i) will have $\frac{N(N+1)}{2}$ cumulative expansions. The last two nodes (e and g) will each be expanded once. Cumulatively there are $2N + 2 = O(N)$ nodes in the graph and there will be $\frac{N(N+1)}{2} + N + 2 = O(N^2)$ total expansions. This is equivalent to the worst case performance of A* variants B and B' with inconsistent heuristics (Martelli 1977; Mérő 1984; Felner et al. 2011). It is unclear if WA* can have worse performance if the edge costs were to grow exponentially, but it would be simple to build weighted versions of algorithms like B or B' to ensure a maximum of $O(N^2)$ expansions in the WA* paradigm.

Summarizing the results here, we provide examples showing that re-openings in WA* can result in arbitrarily large savings or $O(N^2)$ total expansions in a problem with $O(N)$ states. Thus, the choice of whether to use re-openings is going to depend on the properties of a domain. If there are many transpositions, such as in grid maps, the overhead of re-openings can be expensive.

4.3 Re-expansion Policies

Given the potential impact of re-expansions, several policies for dealing with re-expansions are studied. Because re-expansions never occur with a consistent heuristic in OPEN, there are only two contexts in which re-openings can occur: (1) When expanding states in FOCAL, a shorter path to a state in FOCAL may be discovered. (2) When expanding states in OPEN, a shorter path to a state in FOCAL may be discovered.

When a shorter path is found there are three simple policies that can be followed in either context, although more complex policies have been studied (Sepetnitsky, Felner, and Stern 2016). The *re-open* policy always moves states from CLOSED back to FOCAL when a shorter path is found. The

Table 1: The order of expansions in Figure 3(c)

Order	State	f-cost	Order	State	f-cost
1	t_0	12.00	12	b_1	11.33
2	t_1	10.00	13	b_2	9.33
3	t_2	8.00	14	b_3	7.33
4	t_3	6.00	15	b_4	5.33
5	t_4	4.00	16	b_0	12.67
6	b_4	7.33	17	b_1	10.67
7	b_3	8.67	18	b_2	8.67
8	b_4	6.67	19	b_3	6.67
9	b_2	10.00	20	b_4	4.67
10	b_3	8.00	21	e	16.67
11	b_4	6.00	22	g	6.67

update policy updates the g-cost and parent pointers of a state when a shorter path is found, but does not re-open the state by placing it back on FOCAL. The *ignore* policy ignores states that are already found on CLOSED.

An additional policy can be used when a state from OPEN leads to a shorter path in FOCAL. This policy relies on the fact that IOS has an incumbent solution when expanding OPEN. If all states on the incumbent solution path are marked, the search will know when it has updated (reduced) the cost of the incumbent solution by reducing the g-cost of one of the states on the path. In this case, the cost of the incumbent solution can be reduced without re-expanding the path to update all g-costs. The search must simply take note of the reduction in g-cost, and reduce the stored incumbent solution cost by the same amount. This policy is called the *solution-update* policy. Note that if shorter paths are found to several different states on the incumbent solution path, only the maximum improvement can be used for updating the incumbent solution cost. Thus, in Algorithm 1, one of the three policies following line 13 should be used.

4.4 Sequential or Interleaved Expansions

There are two key differences between Optimistic search and A^*_ϵ. First, A^*_ϵ interleaves expansions from OPEN and FOCAL before a solution is found. Second, in A^*_ϵ FOCAL is defined as the subset of states on OPEN which have been proven to be w-optimal. Because of this definition, A^*_ϵ is restricted from expanding FOCAL in the same best-first ordering as WA*. This contrasts with Optimistic search and IOS which are greedy with respect to f_f in FOCAL. Optimistic search only interleaves expansions after a solution is found if it is using the re-open policy.

Is it worthwhile to interleave expansions in OPEN and FOCAL in IOS before finding a solution? There are two primary arguments against this. First, by interleaving expansions the FOCAL search may expand states with g-cost greater than the cost of solution that is found, and then waste effort proving that these states are optimal. This cannot happen in A^*_ϵ because of the way that FOCAL is defined, but it could happen in the simplified FOCAL list used by IOS. Second, IOS can only use the *solution update* policy if a solution is found. If expansions are interleaved before a solution is found this option will not be available. Thus, a full re-open policy would be needed to propagate shorter solutions from OPEN to FOCAL, which may be too expensive in practice. As a result, IOS does not interleave expansions between OPEN and FOCAL before a solution is found.

4.5 Alternate Priority Functions for FOCAL

Given that it may be beneficial for IOS to avoid re-expansions, IOS is only studied with priority functions for FOCAL that do not require re-opening states when an expansion from FOCAL leads to a shorter path to another state on FOCAL. There are currently three options that are independent of the weight used in FOCAL. The first option is to use the WA* priority function. The second and third options are to use the Φ_{XDP} or Φ_{XUP} functions. These priority functions can be dropped into most A* implementations with little or no changes to data structures, so the influence of these priority functions will be a point of study in experimental results.

4.6 Weights in FOCAL

Previous work has suggested using $w_f = 2w - 1$ (Thayer and Ruml 2008). This weight is provided without justification, but in our work on re-expansions (Chen and Sturtevant 2019), we were surprised to derive this same weight in our analysis. In this section we provide a slightly deeper look into this work. We do not yet have a clear justification of the weight. But, we present the connection as an open problem to explain more deeply by the research community.

We begin by considering the isoline (or contour) of all states that have the same priority. In WA*, this would be states with $b = 1/w \cdot g(n) + h(n)$, where b is the value of the priority function for a particular isoline. If we re-write this as a function of y on x/y axes, where the x-axis corresponds to h-costs and the y-axis corresponds to g-costs, then the function becomes $b = (1/w)y + x$ or $y = (b - x)w$. If we measure the slope of this curve by taking the derivative with respect to x, we see that the slope is a constant $-w$. Under this formulation the slope is the negation of the optimality bound. A constant slope means that WA* allows uniform suboptimality from the beginning to the end of the path that is discovered.

The Φ functions introduced by Chen and Sturtevant (2019) change where the suboptimality is allowed along a path. Thus, the slope of the isolines of the priority function change across a path, as seen in Figure 1. From the theoretical assumptions, it is possible to show that slope cannot be larger than -1 and cannot be smaller than $-2w + 1$ for any given $w > 1$. Thus, if one wants to return a solution with suboptimality w_1 and not perform re-expansions, the isolines of the priority function used cannot exceed a slope of $-2w_1 + 1$, which corresponds to a weight of $w_2 = 2w_1 - 1$, the same weight recommended in previous work.

If we analyze the slope of $\Phi_{XDP}(x, y) = \frac{1}{2w}[y + (2w - 1)x + \sqrt{(y - x)^2 + 4wyx}]$ we find that each isoline has the form $y = ((b - x)(bw - wx + x))/b$, where b is some fixed priority of $\Phi_{XDP}(x, y)$. (That is, we substitute b for $\Phi_{XDP}(x, y)$ and then solve for y as above.) For this curve, the slope at $x = b$ is -1 and the slope at $x = 0$ is $1 - 2w$:

Table 2: Node reductions in IOS using the improved termination condition.

Bound	With Bound	No Bound	Gain
1.25	15,402	16,802	9.1%
1.50	10,797	11,219	3.9%
2.00	7,348	7,842	6.7%
3.00	5,425	5,750	6.0%

Table 3: Node reductions in IOS using solution updating.

Bound	Updating	No Updating	Gain
1.25	11,423	15,402	34.8%
1.50	8,715	10,797	23.9%
2.00	6,573	7,348	11.8%
3.00	5,386	5,425	0.7%

$$\frac{\partial y}{\partial x} = (2(w-1)x)/b - 2w + 1|_{x=0} = 1 - 2w$$

Related to this, $\Phi_{XUP}(x, y)$ is the priority function that has a slope of -1 at $x = 0$ and $1 - 2w$ at $x = b$.

Thus, $w_f = 2w - 1$ may work well because it is related to the maximum slope that is allowed by an isoline that maintains w-optimality. It is an open question for researchers to further explain this connection.

4.7 Sketch of the Completeness of IOS

Assuming that the search used to drive expansions in FOCAL is complete on finite problems, then IOS will also be complete. There are three cases to handle: (1) If there is no solution the search can terminate after exhausting FOCAL and return that there is no solution. (2) If the search in FOCAL finds a solution that is more than w-optimal, it will continue to expand OPEN until the optimal solution is found, and then terminate with the optimal solution. (3) Finally, if the search finds a solution that is w-optimal in FOCAL then it will be able to prove this by performing expansions in OPEN. Because the problem is finite, a solution exists, and re-expansions are not necessary inside OPEN, $f_{\min}$ will be increased to a value sufficiently large to prove the quality of the solution before the goal is expanded in OPEN.

5 Experimental Results

Experimental results are broken into two pieces. First, the influence of different parameters is studied on a set of test domains. Then, a broader comparison between different algorithms is studied across new domains.

5.1 Study of Parameters

This section studies the influence of four IOS parameters across four domains. These domains are pathfinding on grid-based benchmarks (Sturtevant 2012) with an octile heuristic, the sliding-tile puzzle with Manhattan distance, the heavy sliding-tile puzzle with heavy Manhattan distance, and the heavy pancake puzzle with a modified GAP heuristic. The heavy sliding-tile puzzle uses linear weights - the cost of moving tile n is n. The heavy pancake puzzle uses the maximum size of the top and bottom pancake in a stack being flipped as the action cost (Gilon, Felner, and Stern 2016).

For grid maps we tested on 15,928 problem instances from the Dragon Age: Origins (DAO) map set. All maps and problem sets were used, except that for each problem bucket on a map we only used a single random instance out of each bucket of 10 problems. In the sliding tile puzzle we used the standard 100 Korf instances. On the pancake puzzle we test on 50 random instances. None of the results compared have significant impact on running time, so only node expansions are reported.

Study of Re-openings The first parameter we study is the impact of re-opening states. Because of the nature of the results, a numerical comparison is not necessary. In the grid pathfinding problems re-opening is catastrophically bad, as it appears that near worst-case performance occurs in practice. This is partially because grid pathfinding has many small cycles with different costs, especially due to the diagonal edge weights. On other domains, allowing re-openings results only has a small impact on average performance, with the exception of the heavy sliding-tile puzzle, where performance is approximately 30% worse with re-openings. Although near-worst-case performance is seen in one domain, the theoretical best-case improvements is never seen. Thus, re-openings are always turned off in further experiments.

Study of Improved Termination Condition Next we look at the impact of the improved termination condition. The improved condition is able to help when there is a local minima around the start state. But, if the state with $f_{\max}$ is the start state, then this condition has no impact. As with re-openings, the improved termination condition has a different impact in grids than in the other domains. On grids there is a small but notable improvement in performance, while in the other domains there is no significant gain. We illustrate this gain using IOS with a standard WA* focal list using $w_f = 2w - 1$ in Table 2. The largest gains in node expansions comes with the smallest optimality bound.

Study of Solution Updating Next we study the impact of the solution-update policy. That is, when re-expansions are disabled, but expansions in OPEN are allowed to shorten the incumbent solution. This reduces the work needed to prove the suboptimality bound and improves solution quality. This parameter has the largest improvement on grid maps with low weights. These gains are presented on top of the gain from the improved termination condition in Table 3.

Re-openings have the potential to be valuable, in that shorter solutions can be propagated through the search space. But, in grid domains this propagation is too expensive. The solution updating approach is a less expensive way of recovering shorter solutions without performing re-openings in practice.

Study of Alternate Priority Functions In the final portion of this study, we look at the impact of the best previous enhancements with different priority functions for FOCAL that avoid node re-expansions. Results are presented across

	DAO Grids			15 Puzzle			Heavy 15-Puzzle			Heavy Pancake		
Bound	Φ_{XDP}	WA*(w_f)	Φ_{XUP}	Φ_{XDP}	WA*(w_f)	Φ_{XUP}	Φ_{XDP}	WA*(w_f)	Φ_{XUP}	Φ_{XDP}	WA*(w_f)	Φ_{XUP}
1.25	**10,940**	11,423	11,695	**264,112**	350,976	521,722	668,032	**397,710**	741,031	-	-	-
1.50	**8,533**	8,715	8,573	**35,280**	47,659	76,756	427,210	**152,310**	174,369	**211,783**	1,074,523	-
2.00	6,829	6,573	**6,266**	**12,050**	14,240	17,397	**64,128**	71,672	85,007	**3,412**	23,567	405,185
3.00	5,818	5,386	**5,060**	**4,468**	4,633	6,627	**48,226**	52,868	48,949	**63**	382	10,342

Table 4: Average performance for IOS with WA*, XDP and XUP as priority functions for FOCAL.

all domains. No algorithms perform re-openings, but solution updating and the improved termination conditions are enabled.

The complete results are in Table 4. For each domain, three different priority functions are used along with 4 different weights. All approaches are able to solve all problems except in the Heavy Pancake puzzle, where some algorithms could not solve problems with lower weights.

IOS with the Φ_{XDP} has the best performance in 11 of the 15 weight/domain combinations tested. The only exception is the Heavy 15-puzzle, where there are a few problems where Φ_{XDP} has very poor performance. We are continuing to study this domain to better understand the performance here.

5.2 Comparison Against Other Algorithms

Experimental results continue with a broader set of algorithms. These include the original version of Optimistic search, Weighted A*, EES, and DPS. We also tested WA* using the new priority functions Φ_{XDP} and Φ_{XUP}. These are compared to IOS with solution updating, no re-openings, and the improved termination condition. Additionally, variants of EES and Weighted A* are used that drop duplicates (denoted by DD in the results.) Note that EES is using a distance function not used by the other algorithms.

The domains used for these tests feature a variety of grid-like domains. The potential duplicate states test the effect of the new strategies towards dealing with the duplicates in each domain. Lifecost and uniform cost grids (Thayer and Ruml 2008), and heavy vacuum world (Thayer and Ruml 2011) are used, along with the racetrack domain (Cserna et al. 2018). The results in each of these domains are found in Tables 5-8.

Each table reports the average number of nodes expanded across a variety of domains and suboptimality bounds. Across the variety of domains tested there is not a single dominant algorithm, although the Φ_{XDP} variants of IOS and WA* consistently perform well. In the life-cost grids, shown in Table 5, WA*(Φ_{XDP}) does well at the lowest suboptimality bound, but then loses out to IOS(Φ_{XDP}). In the heavy version of vacuum world, shown in Table 8, WA*(Φ_{XDP}) outpaces the other algorithms until the higher suboptimality bounds where IOS(Φ_{XUP}) eventually expands the fewest number of nodes. We see an incredibly poor behavior for DPS and both versions of EES in this domain. In the racetrack domain, all of the algorithms have nearly the same behavior except for the EES variants and DPS. EES_DD requires fewer expansions than the rest and performs similarly to EES until higher suboptimality bounds. At those bounds EES performs a large number of re-expansions, while EES_DD can avoid those expansions. Additionally, DPS can, without explicitly avoiding duplicates, expand far fewer nodes than any of the other algorithms. In unit-cost grids WA*(Φ_{XDP}) and IOS(Φ_{XUP}) do well in comparison to the other algorithms, although EES_DD is competitive at the lower suboptimality bounds. However, again eventually at higher weights we see IOS expanding fewer nodes.

Although there is no clear dominating algorithm from these results, using the Φ_{XDP} and Φ_{XUP} priority functions within WA* and IOS can often outperform or equal the current state-of-the-art in suboptimal heuristic search. This is notable because it is a much simpler algorithm to implement.

6 Conclusions and Future Work

This paper develops a new algorithm framework, Improved Optimistic Search (IOS) which is easy to implement, and provides strong performance across a wide variety of domains. IOS can use new recently developed priority functions that avoid node re-expansions to improve performance. IOS improves performance through a novel termination condition, and options for inexpensively updating the cost of the optimal solution without performing re-expansions.

References

Chen, J., and Sturtevant, N. R. 2019. Conditions for avoiding node re-expansions in bounded suboptimal search. *International Joint Conference on Artificial Intelligence (IJCAI)*.

Cserna, B.; Doyle, W. J.; Ramsdell, J. S.; and Ruml, W. 2018. Avoiding dead ends in real-time heuristic search. In *AAAI Conference on Artificial Intelligence*, 1306–1313.

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

Gilon, D.; Felner, A.; and Stern, R. 2016. Dynamic potential search—a new bounded suboptimal search. In *Symposium on Combinatorial Search (SoCS)*, 36–44.

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* 4(2):100–107.

Heusner, M.; Keller, T.; and Helmert, M. 2018. Best-case and worst-case behavior of greedy best-first search. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 1463–1470.

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

	Lifecost Grids									
Bound	DPS	EES	EES_DD	Optimistic	IOS(w_f)	IOS(Φ_{XDP})	IOS(Φ_{XUP})	WA*_DD	WA*(Φ_{XDP})	WA*(Φ_{XUP})
1.50	-	253,967	34,701	125,962,192	576,912	85,544	206,904	238,847	**13,497**	439,782
2.00	15,337	116,912	10,535	1,119,178	576,912	**5,380**	81,004	75,266	7,584	196,721
3.00	9,951	118,531	9,429	106,905	576,912	**4,675**	32,878	21,179	5,380	81,004
10.0	9,659	117,368	9,035	10,384	576,912	**3,972**	7,011	4,962	4,163	12,120

Table 5: Average node expansion in Lifecost Grids

	Uniform Grids										
Bound	DPS	EES	EES_DD	Optimistic	IOS(w_f)	IOS(Φ_{XDP})	IOS(Φ_{XUP})	WA*	WA*_DD	WA*(Φ_{XDP})	WA*(Φ_{XUP})
1.50	2,281,649	3,623,490	1,359,985	2,012,814	1,579,716	1,383,876	2,071,235	1,999,906	1,405,189	**366,634**	1,547,030
2.00	8,221,572	4,167,581	986,207	5,093,657	1,579,716	650,505	748,730	7,653,234	979,017	**294,503**	1,300,070
3.00	4,104,205	2,819,272	525,109	3,030,813	1,579,716	229,641	**222,466**	5,016,731	368,485	258,716	467,512
10.0	2,593,375	2,683,620	360,917	2,209,714	1,579,716	181,722	**162,129**	2,791,159	187,555	196,719	176,647

Table 6: Average node expansion in Uniform 4K Grids

Mérő, 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 Transactions on Pattern Analysis and Machine Intelligence* PAMI-4(4):392–399.

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

Russell, S., and Norvig, P. 2009. *Artificial Intelligence: A Modern Approach*. Upper Saddle River, NJ, USA: Prentice Hall, 3rd edition.

Sepetnitsky, V.; Felner, A.; and Stern, R. 2016. Repair policies for not reopening nodes in different search settings. In *Symposium on Combinatorial Search (SoCS)*, 81–88.

Stern, R. T.; Puzis, R.; and Felner, A. 2011. Potential search: A bounded-cost search algorithm. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 233–241.

Sturtevant, N. R. 2012. Benchmarks for grid-based pathfinding. *Transactions on Computational Intelligence and AI in Games* 4(2):144–148.

Thayer, J. T., and Ruml, W. 2008. Faster than weighted A*: An optimistic approach to bounded suboptimal search. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 355–362.

Thayer, J. T., and Ruml, W. 2011. Bounded suboptimal search: A direct approach using inadmissible estimates. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 674–679.

Valenzano, R.; Arfaee, S. J.; Stern, R.; Thayer, J.; and Sturtevant, N. 2013. Using alternative suboptimality bounds in heuristic search. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 233–241.

Valenzano, R.; Sturtevant, N.; and Schaeffer, J. 2014. Worst-case solution quality analysis when not re-expanding nodes in best-first search. In *AAAI Conference on Artificial Intelligence*, 885–892.

Wilt, C. M., and Ruml, W. 2014. Speedy versus greedy search. In *Seventh Annual Symposium on Combinatorial Search*, 184–192.

Wilt, C. M., and Ruml, W. 2015. Building a heuristic for greedy search. In *Symposium on Combinatorial Search (SoCS)*, 131–139.

	Racetrack										
Bound	DPS	EES	EES_DD	Optimistic	IOS(w_f)	IOS(Φ_{XDP})	IOS(Φ_{XUP})	WA*	WA*_DD	WA*(Φ_{XDP})	WA*(Φ_{XUP})
1.50	1,456,902	**1,016,769**	1,017,895	3,065,595	3,079,499	1,342,418	1,342,418	3,065,595	1,342,418	2,966,089	3,079,390
2.00	1,397,854	**938,666**	1,002,208	3,008,437	3,079,499	1,342,418	1,342,418	3,047,157	1,342,418	2,861,574	3,079,168
3.00	1,274,770	1,047,692	**831,582**	2,914,233	3,079,499	1,342,418	1,342,418	3,008,437	1,342,418	2,539,610	3,078,460
10.0	**204,193**	1,761,331	553,442	1,147,495	3,079,499	1,342,418	1,342,418	2,515,888	1,342,418	313,420	3,046,144

Table 7: Average node expansion in Racetrack

	Heavy Vacuum World										
Bound	DPS	EES	EES_DD	Optimistic	IOS(w_f)	IOS(Φ_{XDP})	IOS(Φ_{XUP})	WA*	WA*_DD	WA*(Φ_{XDP})	WA*(Φ_{XUP})
1.50	31,299,135	5,838,669	5,838,261	4,006,330	5,258,334	3,349,646	3,063,278	3,907,305	1,901,005	**1,128,979**	1,936,663
2.00	30,833,813	5,650,631	5,646,131	4,465,989	4,869,552	1,507,967	1,549,568	3,906,482	1,374,779	**1,032,033**	1,157,513
3.00	28,965,419	5,521,439	5,494,356	5,270,031	4,740,087	939,792	866,622	4,354,842	1,012,301	935,922	**863,028**
10.0	5,918,733	5,207,662	5,143,950	6,107,241	4,723,115	756,434	**626,939**	5,692,748	724,404	811,844	660,827

Table 8: Average node expansion in Heavy Vacuum World