Suboptimal Search with Dynamic Distribution of Suboptimality

Mohammadreza Hami[1], **Nathan R. Sturtevant**[1, 2]

[1] University of Alberta
[2]Alberta Machine Intelligence Institute (Amii)
hami@ualberta.ca, nathanst@ualberta.ca

Abstract

In bounded-suboptimal heuristic search, the aim is to find a solution path within a given bound as quickly as possible, which is crucial when computational resources are limited. Recent research has demonstrated Weighted A* variants such as XDP that find bounded suboptimal solutions without needing to perform state re-expansions; they work by shifting where the suboptimality in the search is allowed. However, the suboptimality distribution is fixed before the search begins. This paper introduces Dynamic Suboptimality Weighted A* (DSWA*), a search framework that allows suboptimality to be dynamically distributed at runtime, based on the properties of the search. Experiments show that dynamic policies can consistently outperform existing algorithms across a diverse set of domains, particularly those with dynamic costs.

Code — https://github.com/nathansttt/hog2/tree/PDB-refactor/papers/DSWA

Introduction

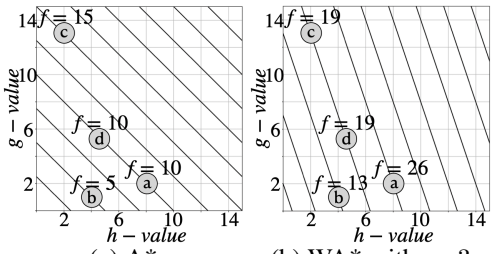
Optimal search algorithms must expand all nodes with an f-value less than C^*, where C^* is the optimal cost to reach the $goal$ from the $start$ state. In harder problems, the number of nodes with $f < C^*$ grows significantly, resulting in increased time to find the optimal solution. This observation has spawned extensive research into suboptimal search methods (Pohl 1970; Pearl and Kim 1982; Thayer and Ruml 2008, 2011; Valenzano et al. 2013; Gilon, Felner, and Stern 2016; Chen and Sturtevant 2021), wherein a suboptimal solution is accepted in exchange for substantial reduction in number of node expansions.

One category of suboptimal search is bounded suboptimal search, which establishes a relationship between the optimal and returned solutions (Gilon, Felner, and Stern 2016). The first historical example of a bounded suboptimal search algorithm is Weighted A* (WA*) (Pohl 1970). It finds solutions that cost no more than a predetermined multiple, w times the optimal solution cost (Ebendt and Drechsler 2009). WA* is a Best-First Search (BFS) algorithm like A* with a different priority function that returns a bounded suboptimal solution. When WA* finds a shorter path to an already expanded state, it has the option of whether or not to re-expand that state with a shorter path (Sepetnitsky, Felner, and Stern 2016). Recent papers have explored BFS algorithms that never re-expand states (BFS-NR) and introduced new priority functions, including the convex downward parabola (XDP) and the convex upward parabola (XUP) (Chen and Sturtevant 2019). XDP prioritizes near-optimal solutions early in the search, allowing for more suboptimal actions as the heuristic value of generated states decreases and the g-value increases. On the other hand, XUP prioritizes near-optimal solutions in states with low heuristic values near the goal, allowing for more suboptimal actions early in the search. Unlike XDP and XUP, WA* distributes suboptimality uniformly across the search. But, in all these priority functions, distribution of suboptimality is fixed before the search begins. Focal list algorithms on the other hand, such as Dynamic Potential Search (Gilon, Felner, and Stern 2016) or Explicit Estimation Search (Thayer and Ruml 2011) adjust their search according to the best path found so far, but have to re-expand states when a shorter path is found and may need to re-sort the OPEN list when the bound on the best path changes.

This paper performs an analysis of the properties of priority functions that can be used with BFS-NR. The primary contribution of this paper is a detailed description of a novel framework for search that dynamically assigns different suboptimality to different parts of the search, while maintaining an overall suboptimality bound, avoiding the requirement to perform re-expansions, and to re-prioritize the OPEN list during search. The suboptimality weight for each part is determined by the policy used in the framework. Several policies are proposed. Although no single policy is uniformly best on all domains, this framework outperforms existing baselines across most benchmarks. It works particularly well on domains with dynamic costs that are not accounted for by the heuristic guidance.

Background

Our problem definition follows closely from Chen and Sturtevant (2021), as we build on their work. It is assumed that all search algorithms are DXBB (deterministic expansion-based black box) (Dechter and Pearl 1985; Eckerle et al. 2017), meaning they are only able to access the state space through a black box expansion function - a state can be accessed only if that state is $start$ or a successor of a



(a) A* (b) WA* with w=3

Figure 1: Different priorities with different priority functions

previously expanded state.

Problem Definition

A Bounded Suboptimal Search (BSS) problem P is characterized by a 5-tuple $P = \{G, start, goal, h, B\}$. The state space, G, is a finite graph that contains states and edges. Each edge $(u, v) \in G$ has a non-negative cost of $c(u, v)$. It is assumed that there is only one directed edge between each pair of states. A path in G is a finite sequence $U = (u_0, ..., u_n)$ of states in G where (u_i, u_{i+1}) is an edge in G for $0 \leq i < n$. The cost of the least cost path from state u to state v in graph G is denoted as $d(u, v)$. The optimal solution cost denoted as C^*, is equal to $d(start, goal)$, the cost of the least cost path from $start$ state to the $goal$. If there is no path from u to v, the value of $d(u, v)$ is infinite. In suboptimal search, the path currently being explored from the $start$ to a state n is not guaranteed to have cost $d(start, n)$.

The heuristic function, denoted as h, estimates the distance between each state and the $goal$. A heuristic is considered *admissible* if for all states n in G, $h(n) \leq d(n, goal)$. A heuristic is *consistent* on an undirected graph if, for any two states m and n, $|h(n) - h(m)| \leq d(n, m)$. If this property holds on a directed graph, we say the heuristic is *strongly consistent*. A heuristic is then *weakly consistent* if for any two states m and n, $h(n) \leq d(n, m) + h(m)$. This paper assumes that the heuristic is strongly consistent.

The final input, $B : \mathbb{R} \rightarrow \mathbb{R}$ is a bounding function. B is used to determine which paths are acceptable solutions, and must satisfy $\forall x \geq 0$, $B(x) \geq x$. BSS is satisficing, so any solution with a cost less than or equal to $B(C^*)$ is acceptable. $B(x) = x$ corresponds to the requirement of finding optimal solutions. In this paper, we exclusively use the function $B_w(x) = wx$, where $w \geq 1$ represents the suboptimality bound for the problem. As we will discuss later, although the overall suboptimality bound is w, this does not prevent the use of a higher suboptimality in certain parts of the search.

Priority Functions for BFS-NR

Algorithm 1 shows the pseudo-code of the Best-First Search algorithm that never re-expands states (BFS-NR). Avoiding re-expansions is useful because re-expansions can significantly add to the cost of finding a solution (Chen et al. 2019). Chen and Sturtevant (2021) provides necessary and sufficient conditions for BFS-NR to find solutions that are bounded by B. We first describe BFS-NR and then provide these conditions.

Algorithm 1: Best-First Search with No Re-expansion

```
1: push(start → OPEN)
2: while OPEN is not empty do
3:    remove state t with minimum priority from OPEN
4:    if t == goal then
5:       return success
6:    end if
7:    push (t → CLOSED)
8:    for each successor s of t do
9:       if s in OPEN and g(s_OPEN) > g(s_SUCCESSOR) then
10:         update g(s) in OPEN to g(s_SUCCESSOR)
11:         update priority of s in OPEN
12:         keep OPEN sorted
13:      else if s is not in CLOSED then
14:         push(s → OPEN)
15:      end if
16:   end for
17: end while
18: return failure
```

BFS-NR uses an open list, where generated states are sorted by their priority value, determining the next state to be expanded. Therefore, the priority function plays a critical role in what path the search returns. One priority function can lead the search to return an optimal path, while another may return a bounded suboptimal path. Since any priority function can be used in BFS-NR, the priority function is not in this pseudo-code. At each step, the state with the minimum priority value is removed from the OPEN and added to CLOSED. The successors of that state are either added to OPEN or, if they are already in OPEN, their new path cost is compared to the cost stored in OPEN. If the cost of the new path is shorter than that in OPEN, the state is updated and it is moved to its correct position to keep the OPEN sorted. States are never removed from CLOSED and put back in OPEN (re-expanded), even if a shorter path is found.

The A* algorithm (Hart, Nilsson, and Raphael 1968) prioritizes states in OPEN using $f(n) = g(n) + h(n)$, where $f(n)$ is the priority of state n and $g(n)$ is the cost of the current cheapest path from $start$ to n in OPEN, while WA* uses $f(n) = g(n) + wh(n)$ where w is the suboptimality bound. Figure 1 shows the f- cost of some pairs of g and h values using A* and WA*. In this figure, pairs that have equal priority fall, by definition, on the same *isoline*. For example, pairs (a) and (d) both have a priority of 10 when using A*. However, switching from A* to WA* changes the isolines and the priority of pairs, even though the same pairs of g and h values are used in both examples. This means that the order in which states are expanded may differ. Using A*, pair (a) is expanded before pair (c). While using WA*, pair (c) has a lower f- cost and is prioritized over pair (a).

This literature also includes various algorithms that require state re-expansions. Focal list algorithms (Pearl and Kim 1982; Thayer and Ruml 2011; Gilon, Felner, and Stern 2016; Fickert, Gu, and Ruml 2022) need state re-expansions,

and some algorithms require re-ordering of the OPEN during the search (Gilon, Felner, and Stern 2016; Fickert, Gu, and Ruml 2022).

Functions with Fixed Distribution of Suboptimality

Chen and Sturtevant (2021) write the priority of a state as $f(n) = \Phi(h(n), g(n))$ and then study the necessary and sufficient conditions of Φ required to avoid re-expansions and still produce a solution bounded by B. We provide the conditions here. Given our assumption that the heuristic is strongly consistent, we use the stronger form of condition 5.

$$\Phi(h+\delta, g) > \Phi(h, g) \quad \text{for } \delta > 0 \quad (1)$$
$$\Phi(h, g+\delta) > \Phi(h, g) \quad \text{for } \delta > 0 \quad (2)$$
$$\Phi(h, g+\delta) \leq \Phi(h+\delta, g) \quad \text{for } \delta > 0 \quad (3)$$
$$\Phi(h, 0) = h \quad (4)$$
$$\Phi(0, B(h)) = h \quad (5)$$
$$\Phi(h+\delta, g+\delta) \leq \Phi(h, g) + 2\delta \quad \text{for } \delta > 0 \quad (6)$$

Given these conditions, a number of different Φ-functions have been derived (Chen and Sturtevant 2019, 2021). $\Phi_{\text{WA*}}$ is functionally equivalent to WA*, but adjusted due to boundary conditions (4 and 5) in the theory.

$$\Phi_{\text{WA*}}(h, g) = \frac{1}{w}g + h$$
$$\Phi_{\text{XDP}}(h, g) = \frac{1}{2w}[g + (2w-1)h + \sqrt{(g-h)^2 + 4whg}]$$
$$\Phi_{\text{XUP}}(h, g) = \frac{1}{2w}(g + h + \sqrt{(g+h)^2 + 4w(w-1)h^2})$$
$$\Phi_{\text{PWXD}}(h, g) = \begin{cases} g + h & \frac{g}{h} < 1 \\ \frac{1}{w}(g + (2w-1)h) & \frac{g}{h} \geq 1 \end{cases}$$
$$\Phi_{\text{PWXU}}(h, g) = \begin{cases} \frac{1}{2w-1}g + h & \frac{g}{h} < (2w-1) \\ \frac{1}{w}(g+h) & \frac{g}{h} \geq (2w-1) \end{cases}$$

It may not be clear how these priority functions work. But, looking at these functions geometrically reveals this more clearly. As $\Phi(h(n), g(n))$ is a function of $g(n)$ and $h(n)$, the priority function is a three-dimensional surface on the h-g plane, where the height of this surface indicates the priority value of a state given its g-cost and h-cost. Since the h-g plane is two-dimensional, we again use isolines to indicate the set of all g/h pairs on the h-g plane that have the same priority. In Figure 1a and Figure 1b, some of the isolines of A* and WA* Φ-functions are drawn as straight black lines.

Figure 2 shows some of the isolines for XDP, XUP, PWXD and PWXU. The slope of the isoline controls the suboptimalty of the search. A slope of -1 corresponds to an optimal search with $w = 1$, while a slope of $-(2w-1)$ corresponds to a search with weight $(2w-1)$. Thus, XDP and PWXD are optimal or near-optimal near the start state (on the h-axis with $g = 0$), and are maximally suboptimal near the goal (on the g-axis with $h = 0$). XUP and PWXU have maximum suboptimality near the start, and are near-optimal or optimal near the goal. We call these functions *fixed Φ-functions*, as they have a fixed distribution of suboptimality regardless of what happens throughout the entire search. PWXD and PWXU are piece-wise functions; on one side of the red ray the search is A*, while on the other side of the red ray, the search is WA* with a weight of $2w-1$.

The Disadvantage of Using Fixed Φ-functions

In this section we show that on some problems there exist bounded-optimal solutions that cannot be returned by the Φ-functions in the previous section, but a custom Φ-function can be built that achieves this.

Consider the problem in Figure 3, with five nodes labeled with their respective heuristic costs and edges labeled with action costs between them. The heuristic is perfect, so the heuristic value of each state is the optimal path cost from that state to the goal. The start state is labeled as S and the goal state as G. The optimal path from S to G with the cost of 160 is through S, A, M, B, and G. Edge M-G is labeled with two costs, creating two scenarios. The second scenario shows a failure of fixed Φ-functions in returning a bounded solution with the fewest number of node expansions.

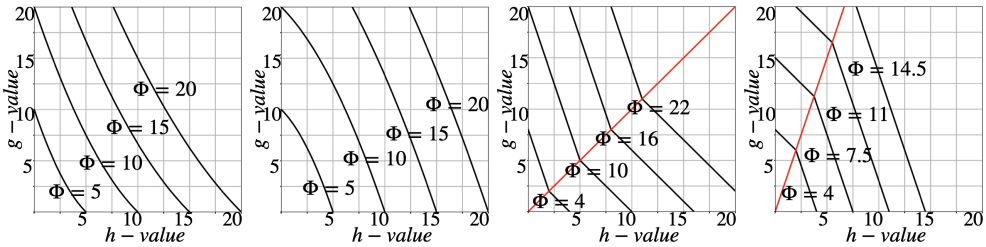
In Figure 3, with $w = 2$, all possible solutions in this graph are bounded, but the path from node S-M-G requires the fewest number of node expansions among all solutions. If we assign a cost of 70 to the M-G edge, among all of the fixed Φ-functions introduced in the previous section, only Φ_{PWXU} returns S-M-G as the solution and performs two node expansions (S and M; the search terminates before expanding G). However, if we change the cost of the M-G edge to 150, the S-M-G path with a total cost of 320 will not be returned by any of these functions. One Φ-function[1] that can solve this problem with two node expansions is:

$$\Phi_{\text{x}} = \begin{cases} g + h & \frac{g}{h} \leq \frac{50}{110} \\ [g + (2w-1)h]\frac{8}{11w-3} & \frac{50}{110} < \frac{g}{h} \leq \frac{170}{70} \\ [g + h]\frac{14w+10}{33w-9} & \frac{170}{70} < \frac{g}{h} \leq \frac{200}{40} \\ [g + (2w-1)h]\frac{14w+10}{11w^2+19w-6} & \frac{g}{h} > \frac{200}{40} \end{cases}$$

In this problem WA* generates A with $f(A) = 50 + 2 \cdot 110 = 50 + 220 = 270$ and $f(M) = 170 + 2 \cdot 70 = 310$, expanding A before M. But Φ_{x} will generate A with $\Phi_{\text{x}}(110, 50) = 160$ and M with $\Phi_{\text{x}}(70, 170) = [170 + 3 \cdot 70] \cdot 8/19 = 160$. If ties are broken towards the largest g-cost, M will be expanded first. Now B will be generated with $\Phi_{\text{x}}(40, 200) = [200 + 40] \cdot 2/3 = 160$, and G will be generated with $\Phi_{\text{x}}(0, 320) = [320] \cdot 1/2 = 160$, and the goal can be chosen for expansion, finding a bounded-suboptimal path with only two expansions.

Φ_x solves *this* problem in two expansions, but we can create a family of variants where it would fail on all but this problem, and a different Φ-function would succeed. In this paper we show how to create dynamic Φ-functions that can adapt to search conditions to provide better performance while avoiding state re-expansions. A variant of the greedy policy, introduced later in the paper, can solve this class of similar problems with minimal node expansions.

[1] Note that $\Phi_x(h, g)$ is only designed for $w = 2$.



(a) Convex Downward Parabola (XDP) (b) Convex Upward Parabola (XUP) (c) Piece-wise Convex Downward (PWXD) (d) Piece-wise Convex Upward (PWXU)

Figure 2: An example of different Φ-functions producing different isolines for $w = 2$

Dynamic Φ-functions

By dynamically defining a Φ-function, we can adjust where in the search the suboptimality is allowed. Our goal is to minimize the number of node expansions while returning bounded-optimal solutions. To achieve this, we need to understand the structure of a BFS-NR search to see where dynamic decisions about suboptimality can be made.

In particular, we observe that, when plotted on a h-g plane, the search proceeds incrementally across the state space, suggesting that we can define Φ incrementally. This is demonstrated in Figure 4a, where each point corresponds to the g-value and h-value of a state expanded by WA* in a given problem. The search starts at the h-axis with $g = 0$ and expands states to reach the *goal* state with on g-axis with $h = 0$. States that were expanded later in the search are represented by points with darker colors. If we break the h-g plane into different regions, the Φ-function only has to be defined in a region once that region is reached by the search.

Figure 4b shows an example of how rays and regions are defined on this plane. Every line that connects a point on the plane to the point $[0, 0]$ is called a *ray*. The term *slope of a state* refers to the slope of the ray created by that state. The area that lies between between adjacent rays, is called a *region*. This paper is based on the observation that the Φ-function does not have to be defined in region$_i$ until a state is generated inside that region. So, it is possible to expand the start state, and only then begin to determine the Φ-function.

Range of Possible Isolines

In order to build dynamic Φ-functions, we first assume that the Φ-function is piece-wise, with each piece defined as $k_i(g + w_i \cdot h)$, where k_i is a normalizing constant that ensures that the Φ-function is continuous. We then define the range where the isoline of the function can be found, assuming $B_w(x) = w \cdot x$. We build this range from the definitions of PWXD and and PWXU, which trace the outer boundaries of the region.

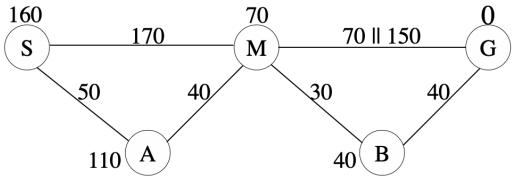
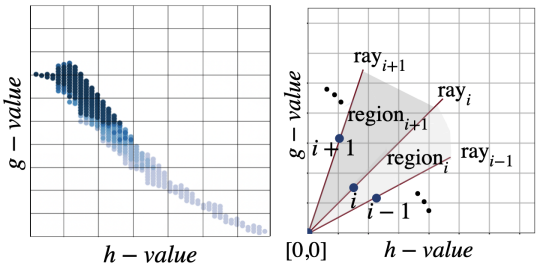


Figure 3: An example problem where a path is bounded, but cannot be returned by existing Φ-functions.



(a) Search Direction (b) Rays and Regions

Figure 4: The idea of creating a dynamic Φ-function

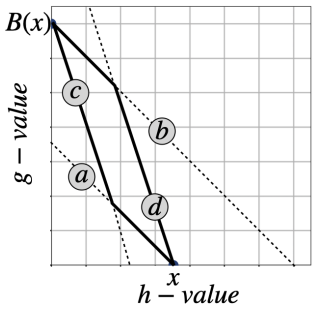


Figure 5: The never re-open region

Let the *never re-open region* for an isoline of priority x be a parallelogram bounded by four lines. Figure 5 shows the never re-open region on h-g plane. The first line, labeled (a), is with slope -1 that intersects $(x, 0)$. The second, labeled (b), is with slope -1 that intersects $(0, B(x))$. The third, labeled (c), is with slope $(1 - 2w)$ that intersects $(0, B(x))$. The last, labeled (d), is with slope $1 - 2w$, that intersects $(x, 0)$.

Lemma 1. *If all isolines for a Φ-function stay inside their never re-open region with slope between -1 and $(1 - 2w)$, the Φ-function will not require re-expansions in order to be bounded.*

Proof. First consider line (a) in Figure 5. The isoline starting from the h-axis can only cross below this line if Condition 3

is violated, meaning the isoline has slope > -1, or equivalently $w_i < 1$. Thus, it is necessary for the isoline to stay on or above (a). In general, for the same reason, at any point in the curve the isoline must have slope ≤ -1, or, equivalently, $w_i \geq 1$. Finally, the isoline cannot go above line (b), as doing so would again require the line to have slope ≤ -1 in order to reach the g-axis at the point required by Condition 5. This establishes the upper-limit on the slope, or, equivalently, the lower-limit on w_i for any segment.

Second, consider the final segment of any piece-wise isoline of the form $k_i(g+w_i \cdot h)$. When $h = 0$ and $B(h) = wh$, Condition 5 must hold, giving:

$$\begin{aligned} \Phi(0, B(h)) &= h \\ \Phi(0, wh) &= h \\ k_i(wh + 0) &= h \\ k_i &= 1/w \end{aligned}$$

This mean that every isoline ends with a segment of the form $1/w(g + w_i \cdot h)$. For a segment with this property, we can then check the allowed values for w_i if Condition 6 is to hold.

$$\begin{aligned} \Phi(h + \delta, g + \delta) &\leq \Phi(h, g) + 2\delta \\ 1/w((g + \delta) + w_i \cdot (h + \delta)) &\leq 1/w(g + w_i \cdot h) + 2\delta \\ (g + \delta + w_i \cdot h + w_i \cdot \delta) &\leq (g + w_i \cdot h) + w \cdot 2\delta \\ (\delta + w_i \cdot \delta) &\leq w \cdot 2\delta \\ (1 + w_i) &\leq w \cdot 2 \\ w_i &\leq 2w - 1 \end{aligned}$$

Thus, at the end of the isoline, the last segment must have weight $\leq 2w - 1$, and thus slope $\geq 1 - 2w$.

Finally, to see whether line (d) impacts the isoline, consider the general equation of a piece-wise segment $k_i(g + w_i \cdot h)$ that does not terminate on the h-axis. Assume k_i is chosen so that the isolines are continuous (a necessary assumption), and that the first segment has $k_0 = 1/w_0$ (to meet Condition 4). Then Condition 6 requires:

$$\begin{aligned} \Phi(h + \delta, g + \delta) &\leq \Phi(h, g) + 2\delta \\ k_i((g + \delta) + w_i \cdot (h + \delta)) &\leq k_i(g + w_i \cdot h) + 2\delta \\ (g + \delta + w_i \cdot h + w_i \cdot \delta) &\leq (g + w_i \cdot h) + 2\delta/k_i \\ (\delta + w_i \cdot \delta) &\leq 2\delta/k_i \\ (1 + w_i) &\leq 2/k_i \\ w_i &\leq 2/k_i - 1 \end{aligned}$$

To understand this constraint, we re-write the Φ-function specifically for the isoline when $\Phi = 1$. This gives:

$$\begin{aligned} k_i(g + w_i \cdot h) &= 1 \\ g + w_i \cdot h &= 1/k_i \\ g &= -w_i \cdot h + 1/k_i \end{aligned}$$

This last term is the equation of a line with slope $-w_i$ and intercept $1/k_i$. So, the requirement following from Condition 6 ($w_i \leq 2\frac{1}{k_i} - 1$) puts a limit on the relationship between the slope of the isoline and the g-intercept. This holds for any point in the never re-open region for any $1 \leq w_i \leq 2w - 1$, and thus staying inside line (d) will also guarantee that the search is bounded. □

Staying inside the never re-open region is sufficient, but not necessary. Larger weights than $2w - 1$ can be used before the last segment of a piece-wise isoline. But, practically speaking, Φ-functions like XUP do not have strong performance on most domains, so here we chose to limit the maximum weight to $2w - 1$.

Algorithm 2: DSWA*

```
1:  push ((0, 1) → data[ ])
2:  push (start → OPEN)
3:  while OPEN is not empty do
4:     remove state t with minimum Φ-value from OPEN
5:     if t == goal then
6:        return success
7:     end if
8:     push (t → CLOSED)
9:     UpdateRegions(t)
10:    for each successor s of t do
11:       if s in OPEN and g(s_OPEN) > g(s_SUCCESSOR) then
12:          update g(s) in OPEN to g(s_SUCCESSOR)
13:          update Φ of s in OPEN
14:          keep OPEN sorted
15:       else if s is not in CLOSED then
16:          push(s → OPEN)
17:       end if
18:    end for
19: end while
20: return failure
```

Dynamic Suboptimality Weighted A*

Dynamic Suboptimality Weighted A* (DSWA*) is an algorithm that dynamically creates a Φ-function with different weights assigned to each region. When calculating the Φ-value of a state n, the algorithm determines the region i that contains n, and uses the corresponding w_i to return $k_i(g(n)+w_i \cdot h(n))$, where g and h are the values of state n, and k_i is a normalizing constant that ensures the isolines are contiguous. DSWA* uses the slopes of the generated states, and additional information if necessary, to assign a weight to each region. During the search, DSWA* may see states in any region at any time. Due to the different weights in each region, the final path will have varying degrees of suboptimality that can changes dynamically based on the problem instance. In the following, w_i denotes the weight of region i, and w denotes the bound of the suboptimal search problem.

Algorithm 2 presents the pseudo-code for DSWA*. Its structure is similar to BFS-NR: it pushes the $start$ to the OPEN, and while OPEN is not empty, removes the state t with the minimum Φ-value from OPEN, checks if it is the $goal$, and appends it to CLOSED (Lines 3-9). Then, for each successor of t, it computes the Φ-value of that successor, and updates its cost if it is already on OPEN, or otherwise appends it to OPEN (Lines 10-18). In the primary additional

Algorithm 3: GetPhi(state s)

```
1: find r_th element of data[ ] such that:
2:    data[r − 1].slope < slope(s) ≤ data[r].slope
3: return k_r(g(s) + data[r].weight · h(s))
```

Algorithm 4: UpdateRegions(state t)

```
1: find successor m of t that has the largest slope
2: if slope(m) > data.back.slope then
3:    w_min, w_max ← GetWeightRange(slope(m))
4:    w_next ← Policy(w_min, w_max, ...)
5:    push ((slope(m), w_next) → data[ ])
6: end if
```

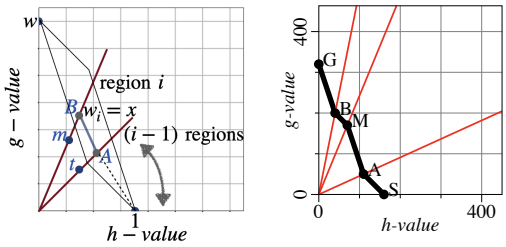
step (Line 9), DSWA* checks the successors of t to see if it is necessary to create a new region and assign it a weight

DSWA* dynamically subdivides the space into an arbitrary number of regions with different weights. This information is stored in a data structure $data[\,]$, where element i of $data[\,]$ stores the information of region i, and includes the range of slopes covered by region i and the corresponding weight used in that region. Elements in $data[\,]$ are sorted based on their slope, with the element having the largest slope placed at the end. Algorithm 3 presents the pseudocode for GetPhi function that calculates the Φ-value of state s. It finds the region r in $data[\,]$ that includes the slope of s, uses the weight stored for that region w_r, and returns $\Phi(h(s), g(s)) = k_r(g + w_r \cdot h)$. Finding the region r for a state s can be done by linear scan, binary search, or, if the rays constrained to be equidistant, by direct lookup.

After appending the expanded state t to CLOSED, DSWA* calls the UpdateRegions function. This function calculates the slope of each successor of t by dividing its g-value by its h-value. It then identifies the successor m with the maximum slope among all successor of t, and compares its slope to the slope of last element in $data[\,]$ ($last$.slope). If the slope of m is greater than the $last$.slope, we have generated a state that is not in any of the previously defined regions. Therefore, a new region and its weight must be set.

DSWA* uses the *GetWeightRange* function to determine the valid range of weights (w_{min} and w_{max}, minimum and maximum valid weight) to choose the next w_i. For any w_i such that $w_{min} \leq w_i \leq w_{max}$, this function has two constraints: First, it should satisfy $1 \leq w_i \leq 2w - 1$. Second, w_i should not cause the isoline to go outside the never reopen region. Then, DSWA* uses a *policy* to determine the appropriate w_i for that region. The policy can use any available information during the search. This includes the number of node expansions in each of the previous regions, the Φ-value of the last expanded state, the h-value and g-value of the expanded state, the heuristic value of any two arbitrary states, or any other relevant observations. Once the weight of the new region is determined, DSWA* updates the $last$ with the slope of the new region and its weight and appends it to $data[\,]$. The priority of all successors is now defined, so the search can continue.

Figure 6a shows the process of creating the new region i



(a) Creation of region i (b) $\Phi = 160$ isoline (Fig. 3)

Figure 6: Examples of how DSWA* works

by DSWA*. As the state t is removed from OPEN, there must be $i - 1$ defined regions. Assume the isoline is defined from point 1 on the h-axis to an arbitrary point A on the plane. When the successor m with the largest slope is generated, we have the ray i and must define the region i and its w_i. If the policy returns x as w_i, the isoline is extended with a line with a slope of $-x$ from A to B connecting the two rays $i-1$ and i. The normalizing constant k_i is used to ensure the isoline is contiguous with the isoline in the previous region. Figure 6b shows the isoline for Φ_x used in Figure 3. Each ray passes through a state in the example; w_i is either 1 or $2w - 1$, depending on the region.

DSWA* Weight Policies

The success of DSWA* is determined by the strength of the policy. We outline several candidate policies.

Greedy Policy If we want the search to continue greedily, we can build a *Greedy Policy* that attempts to assign the successor with the maximum slope the same Φ-value as its parent. This policy will typically place newly generated successors at the top of OPEN (since they are on the same isoline and have thee same f-cost), and thus attempt to greedily move towards the goal. However, if a local minimum is reached where there are no children, or the children cannot be generated with the same Φ as the parent, the search will they have to return to states much earlier in the search before making further progress. For instance, in a random map this would essentially run A* until the first obstacle is hit. The search is then forced to find the optimal way around the obstacle, which is expensive, where WA* would easily move around the obstacle. Thus, this policy does not perform well in practice, although a variant of the policy can solve the class of problems illustrated in Figure 3 with two expansions.

Half Edge Drop Policy We observe that in WA*, a small drop in heuristic value results in a larger drop in priority, because the heuristic is weighted. This gives room for the search to move around small local minima without raising the priority high enough to require expanding previous states in the search. To decrease the Φ-value over time, the *Half Edge Drop Policy* (HEDP) attempts to reduce the priority of a new state by half the cost of the edge from the parent. This approach outperforms the greedy policy, but was opt as strong as other proposed policies.

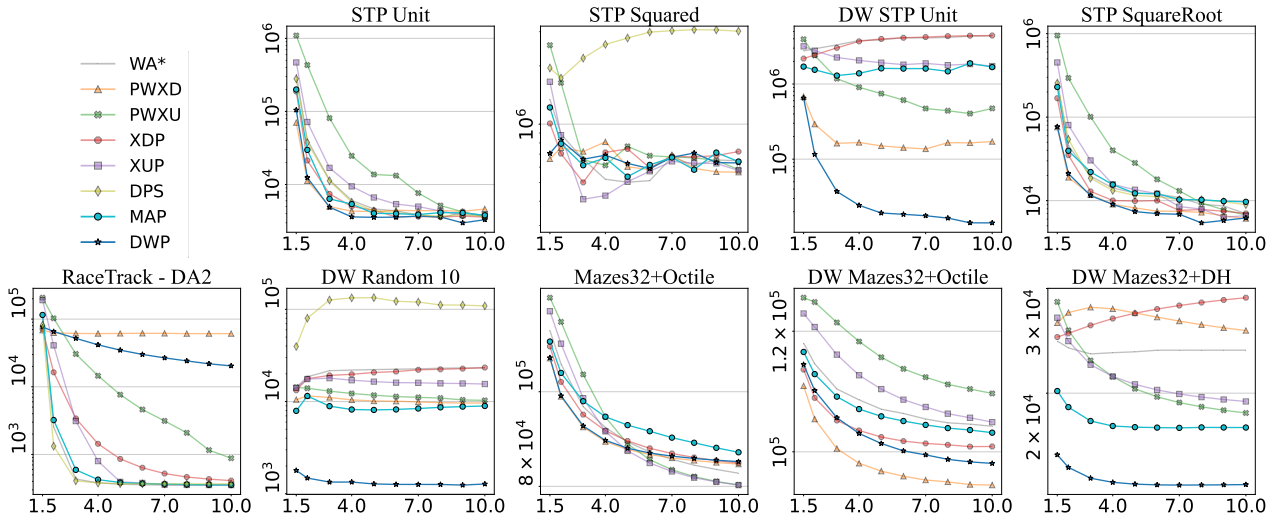


Figure 7: Average number of node expansions to find solutions within a given suboptimality bound

Dynamically Weighted Policy (DWP) Many algorithms differentiate between the number of actions needed to reach the goal and the cost of reaching the goal (Thayer and Ruml 2011). When the search encounters a region of state with weighted action costs, search tends to stall, not expanding state with large g-values due to their large Φ-values. Consequently, the search must expand numerous states from OPEN before eventually expanding the costly states. If this costly region lies along the path to the $goal$, the search either expands many extra states to pass through the region or expand extra states to detour around it. However, by using a policy that increases the search suboptimality (weight), when facing these regions, the search can pass through this region more efficiently. The Dynamically Weighted Policy (DWP) compares $h(t, m)$, to the action cost of the generating successor m from its parent t. If the search is not in a weighted region and the heuristic is accurate, this ratio is equal to one. Then, the policy calculates the angle between the new ray and h-axis, and returns $w_{min}+(w_{max}-w_{min})\cdot(angle/90)^i$ for $i = 3$ (a tuned parameter). This creates an isoline with suboptimality that slowly increases as the search proceeds toward the $goal$ (with $h = 0$). If the search is in a weighted region, the policy returns w_{max}, to facilitate passing through the weighted region efficiently. Note that while our original problem definition supplies a heuristic for the goal, DWP requires a heuristic that can be used between pairs of states (Shperberg et al. 2024).

Moving Average Policy (MAP) The Moving Average Policy (MAP) determines w_i based on the number of node expansions in previous regions. It takes the weighted moving average (wma) of the number of expanded states in the last three regions (where the most recent region has the highest weight), and normalizes the value of wma to be between 0 and 1 (wma_N). Then, MAP defines $w_{low} = \frac{w_{min}+w}{2}$ and $w_{high} = \frac{w_{max}+w}{2}$, and uses them to return $w_{low} + (w_{high} - w_{low}) \cdot wma_N$ as the weight of the new region. We say the search is *progressive* if wma shows the majority of the states are in the most recent region created. In the case of the search not being progressive, MAP returns larger weights, and otherwise, it returns smaller weights to leave room for later local minima and hard situations. For instance, if the number of expansions in the last three regions are 15, 10, and 20 (with 15 being the number of states in the most recent created region), $wma = \frac{3*15+2*10+1*20}{6}$, $wma_N = 0.75$, and returns $w_{low} + (w_{high} - w_{low}) \cdot 0.75$.

Experimental Evaluation

We evaluate DSWA* in a variety of domains and against a variety of algorithms. The experiments were run on a cluster with Intel E5-2683 CPUs and our implementation is in C++ in the HOG2 repository. Algorithms were executed until they found a solution or expanded 10^7 nodes.

The DWP policy is designed to perform well in domains with regions of states that have higher action costs compared to others. We predict that dynamically adjusting the weight when encountering these regions can avoid extra node expansion, but this may not do well if there are no dynamic costs in the problem. MAP is designed to work well in domains with obstacles and local minima. However, we do not know if increasing the number of obstacles and local minima will impact the performance of this policy. To gain a better understanding of these aspects, we conducted an empirical evaluation across domains with different characteristics.

Classic Search Domains Our study included grid maps from the MovingAI pathfinding repository (Sturtevant 2012) using the octile heuristic. We tested our policies on all maps from the Dragon Age 2 (DA2) game, as well as on all artificial maps such as Random maps and Mazes maps. In the 8-connected grid domains, we used the octile heuristic function. Additionally, we evaluated the standard 100 Korf instances of the Sliding Tile Puzzle (STP) (Korf 1985) using the Manhattan Distance (MD) heuristic. In addition to unit cost STP, our experiments include the STP Squared and

STP SquareRoot variants. In STP Squared, the cost of moving the tile x is equal to x^2. In STP SquareRoot, the cost of moving the tile x is equal to $sqrt(x)$. We also examined the RaceTrack domain (Sutton and Barto 2018). We used DA2 maps for RaceTrack, and evaluated at most 1000 problems for each map, sampled randomly from all problems. The RaceTrack heuristic calculates the shortest distance between a state and the goal, and counts the number of actions needed to cover this distance, taking into account the maximum speed at that state.

Dynamically Weighted Domains The action costs in these domains are uniform across the state space and are captured by the default heuristics. However, many applications, such as pathfinding in video games, have different costs depending on the terrain (Sturtevant et al. 2019).

To study these cases, and show the importance of using a dynamic policy, we design dynamically weighted (DW) problems where the cost of actions in a region of the space are modified. The location of these modified costs is not known by the heuristic function. In grid map domains such as mazes and random10, we modify the cost of a range of rows or columns of states randomly between the $start$ and the $goal$. In STP with unit costs, we select a random value v, where $10 \leq v \leq h(start) - 10$, and modify the cost of any state n where $v - 5 \leq h(start, n) \leq v + 5$. The weighted actions have their costs multiplied by $2w-1$. Full details are described in (Hami 2025), and can be found in the code.

Baseline Algorithms We compare against several baseline algorithms. Algorithms that do not need re-expansions include WA* (Pohl 1970), and BFS-NR with Φ functions XDP, XUP, PWXD and PWXU. We also compared against Dynamic Potential Search (DPS) (Gilon, Felner, and Stern 2016). We performed some comparisons against RR-d (Fickert, Gu, and Ruml 2022), however a significant difference between the description and published implementation prevented further comparisons until that is resolved, something we have discussed with the authors of that paper.

Analysis of Results

Figure 7 shows experimental results across different domains. Each plot shows the average number of node expansions required to find solutions on the y-axis in logarithmic scale, within specific suboptimality bounds on the x-axis. Hami (2025) has additional results along with numerical tables with confidence values, and overall rankings.

Robustness of Algorithms In our experiments across different domains with various properties, none of the baselines consistently perform well across all domains when compared to our framework. For instance, Φ_{PWXD} performs well in STP but poorly in RaceTrack, and Φ_{XUP} struggles in STP and excels in RaceTrack. In DW Random 10, MAP is the second best algorithm after DWP, and the closest baseline to it is PWXD. But in RaceTrack PWXD fails and MAP is still very close to the best algorithm. We see that the performance of all algorithms is affected by suboptimality bounds, with some performing well under small bounds but performing worse than others as the suboptimality bound increases. DSWA* on the other hand shows robust performance across all domains and suboptimality bounds. In almost all domains, its ranking among all algorithms rarely changes as the suboptimality bounds increases. Also, all baselines fail compared to DSWA* in dynamically weighted domains.

Dynamically Weighted Domains The DW domains show the potential of DSWA*. To reduce the number of node expansions, DSWA* needs to be able to dynamically recognize when larger weights are helpful, which is easy in the DW domains. In DW Random 10% and DW STP, DSWA* performs almost ten times better than the best baseline across all suboptimality bounds, results that are significant with 95% confidence. In DW Mazes32+Octile however, we do not observe the same performance, which we discuss next.

Heuristic Accuracy DSWA* can dynamically adjust suboptimality during runtime based on properties of the state space. However, if the search is stuck in a local minimum and not making progress, it cannot adjust weights to escape after getting stuck. In particular, if the search find a state where the heuristic estimate is very low, but the actual distance from the goal is large, nearly the entire Φ-function will be defined and will no longer be able to be adjusted.

Thus, we hypothesized that the octile heuristic is too weak for maze maps, and tested this by using a differential heuristic (DH) (Sturtevant et al. 2009) instead of the octile heuristic. The results confirm our hypothesis. Using the DH with 10 pivots on DW maze maps, both MAP and DWP significantly outperformed other baselines. This suggests that DSWA* well-suited for domains with unpredictable costs, but less so for local minima with very low heuristic costs.

Running Time The implementation of DSWA* used for the experiments presented here uses a linear scan to find the correct region for each state generated. Additional experiments have been run with an alternate implementation that fixes the size of the each region (e.g. $0.5°$) in order to do constant-time lookups. In DW STP there is almost no difference in runtime between the two implementations and other algorithms; speed is determined by total expansions.

In grid domains, the constant-time region lookups reduce the rate of expansions from 50% slower than the baseline algorithms to 20% slower. The gains from DSWA* outweigh the overhead. Additional profiling and optimizations are expected to further reduce this gap. We note, however, that using constant-size regions impacts the MAP policy. It is a matter of future work to study this relationship further, as well as to further optimize the implementation.

Conclusions

This paper introduced the DSWA* algorithm with policies that can dynamically choose where to be suboptimal at runtime, while avoiding state re-expansions. DSWA* is shown to be robust across domains and to work particularly well in domains with dynamic weights.

Acknowledgments

The initial ideas in this paper were inspired by research conversations between the second author and Jingwei Chen.

Shaoyu Tang contributed the piece-wise approximation of XDP used in the DWP policy. This work was supported by the National Science and Engineering Research Council of Canada Discovery Grant Program and the Canada CIFAR AI Chairs Program. This research was enabled in part by support provided by Prairies DRI and the Digital Research Alliance of Canada (alliancecan.ca).

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)*.

Chen, J.; and Sturtevant, N. R. 2021. Necessary and sufficient conditions for avoiding reopenings in best first suboptimal search with general bounding functions. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, 3688–3696.

Chen, J.; Sturtevant, N. R.; Doyle, W.; and Ruml, W. 2019. Revisiting Suboptimal Search. *Symposium on Combinatorial Search (SoCS)*, 18–25.

Dechter, R.; and Pearl, J. 1985. Generalized best-first search strategies and the optimality of A. *Journal of the ACM (JACM)*, 32(3): 505–536.

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

Eckerle, J.; Chen, J.; Sturtevant, N.; Zilles, S.; and Holte, R. 2017. Sufficient conditions for node expansion in bidirectional heuristic search. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 27, 79–87.

Fickert, M.; Gu, T.; and Ruml, W. 2022. New Results in Bounded-Suboptimal Search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, 10166–10173.

Gilon, D.; Felner, A.; and Stern, R. 2016. Dynamic potential search—a new bounded suboptimal search. In *Proceedings of the International Symposium on Combinatorial Search*, volume 7, 36–44.

Hami, M. 2025. *Suboptimal Search with Dynamic Distribution of Suboptimality*. Master's thesis, University of Alberta.

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.

Korf, R. E. 1985. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial intelligence*, 27(1): 97–109.

Pearl, J.; and Kim, J. H. 1982. Studies in semi-admissible heuristics. *IEEE transactions on pattern analysis and machine intelligence*, (4): 392–399.

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

Sepetnitsky, V.; Felner, A.; and Stern, R. 2016. Repair policies for not reopening nodes in different search settings. In *Proceedings of the International Symposium on Combinatorial Search*, volume 7, 81–88.

Shperberg, S. S.; Felner, A.; Siag, L.; and Sturtevant, N. R. 2024. On the Properties of All-Pair Heuristics. *Symposium on Combinatorial Search (SoCS)*.

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

Sturtevant, N. R.; Felner, A.; Barer, M.; Schaeffer, J.; and Burch, N. 2009. Memory-based heuristics for explicit state spaces. *International Joint Conference on Artificial Intelligence (IJCAI)*, 609–614.

Sturtevant, N. R.; Sigurdson, D.; Taylor, B.; and Gibson, T. 2019. Pathfinding and Abstraction with Dynamic Terrain Costs. In *Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*.

Sutton, R. S.; and Barto, A. G. 2018. *Reinforcement learning: An introduction 2nd edition*. MIT press Cambridge.

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 *International Joint Conference on Artificial Intelligence (IJCAI)*, volume 2011, 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.