Single Agent Search Course _ project

Replication of –Revisiting Suboptimal Search– paper

Mohammadreza Hami
Department of Computing Science
University of Alberta, Canada
hami@ualberta.ca

Abstract

Suboptimal search algorithms refer to a group of algorithms designed to find a solution faster, rather than the optimal solution. In certain practical applications, such as video games, it is often more important to find a solution quickly than to find the optimal solution. Despite their practical usefulness, the implementation of the previously presented algorithms in this area is often challenging due to their dependence on special heuristics or data structures. This paper addresses these challenges by presenting a new framework named Improved Optimistic Search, which simplifies the implementation process and improves the performance compared to previous algorithms. In addition to introducing this framework, we conduct several studies to evaluate different strategies used within the framework. In addition, one other contribution is evaluating different strategies within the framework to further optimize its performance.

1 Introduction

In optimal search algorithms, to find the optimal solutions the search has to expand all the nodes with a cost of less than C^* meaning the optimal cost to reach the goal. Thus, as the search space becomes larger, the time required to find the optimal solution increases correspondingly. This motivates a broad literature on suboptimal search, which allows for a less-than-optimal solution in exchange for reduced search time significantly.

Weighted A* (WA*) (Pohl 1970) is the classic algorithm for the suboptimal search which finds a solution with a cost of at most w times the cost of the optimal solution. It is one of the most well-known and straightforward suboptimal search algorithms is the bounded suboptimal search algorithm. Bounded suboptimal algorithms are designed with a constraint relation between the cost of the solution found by the search and the cost of the optimal solution for the given problem. (Gilon, Felner, and Stern 2016).

This paper studies the simple w-bounded suboptimal search with a single heuristic function, by building a framework on algorithms like WA^* and Optimistic search (Thayer and Ruml 2008) named Improved Optimistic Search (IOS). Furthermore, we present a series of studies focused on improving termination conditions and solution updating policies, which are two critical components of the algorithm. In the final study, we provide experimental results that compare different priority functions that can be utilized within the algorithm.

The Improved Optimistic Search (IOS) algorithm employs a Focal list to search for a solution and an Open list to ensure that the cost of the solution is within the desired range. In contrast to Optimistic Search, IOS can also utilize a second termination condition to prove that the solution found is w-bounded without the need to expand nodes from the Open list. Additionally, we explore a broader class of priority functions (Chen and Sturtevant 2019) that are compatible with the proposed framework and have the potential to improve the algorithm's performance compared to the WA^* priority function.

To summarize, the IOS framework offers several advantages, including the ability to reduce the number of re-expansions required during the search process. Additionally, the framework can use a variety of priority functions for its Focal list, which has been shown to significantly impact the algorithm's performance. Furthermore, the IOS framework can employ an improved termination condition that helps to terminate the search faster and prevent unnecessary expansions from the Open list, further increasing the efficiency of the algorithm.

2 Background and Related Work

Best-first search algorithms use an Open list in which the generated nodes are sorted according to their $f(n)$. In these algorithms, at each step, the node with the minimum f is expanded from the Open list and its children are appended to it. In A^*, $f(n) = g(n) + h(n)$ meaning that the f cost of each node is equal to the cost of reaching that node from the start state, plus the cost of reaching the goal state from that node. Thus, f cost of any node n, is an estimate of a solution cost which has n as one of the nodes between the start and the goal state. Therefore, as we expand the node with the minimum f cost, we are expanding the best node which can lead us toward the optimal solution.

In WA^* on the other hand, the priority function that is used is $f(n) = g(n) + w \cdot h(n)$ and it returns the w-optimal solutions meaning the solutions that their cost is not more than w times more than the optimal solution. WA^* is one of the most well-known and used algorithms since it has good performance and also it is very easy to implement.

In general, we use a function $\Phi(x, y)$ as the priority function in the best-first search algorithms. Thus, we can define $f(n) = \Phi(h(n), g(n))$ such that in A^*, $\Phi(h(n), g(n)) = h(n) + g(n)$ and in WA*, $\Phi(h(n), g(n)) = w \cdot h(n) + g(n)$. Later in this section, we will discuss other priority functions which can be used in the search algorithms.

WA* handles both finding the solution and proving that the found solution cost is w-optimal using its Open list. On the other hand, a large number of bounded suboptimal search algorithms have been proposed that can be viewed as an implementation of a general framework called a Focal list (Pearl and Kim 1982). The idea is to separate the task of finding the solution and proving it is w-optimal and handling each task using one of the Open list and Focal list.

Open list is sorted based on $g(n)+h(n)$. As this is the estimate of the optimal solution, we can use this list to check whether the solution is w-optimal. Focal list on the hand, is sorted based on another Φ and is used to just find a solution. Once the w-optimal solution is found using the Focal list, we will re-expand the required nodes from Open to prove desired amount of optimality. In following chapters, we will discuss the Φ functions in more details and in Section 5 we will demonstrate an experimental result of comparing different Φ functions on Focal list.

The first algorithm that used the Focal list idea was A^*_ϵ (Pearl and Kim 1982). It was devised to equip A^* with such completion capabilities while still guaranteeing that the solution found at termination would not exceed the optimal one by a factor greater than $1 + \epsilon, (\epsilon \geq 0)$. It expands nodes from Focal if and only if it is proven that the cost of that node is within the suboptimality bound. Also, it only allows re-expansions whenever a shorter path is found to a state that was expanded before.

Optimistic Search (Thayer and Ruml 2008) is another suboptimal search algorithm that uses a Focal list in search. The main advantage of this algorithm over the prior one is that nodes expanded from Focal are not required to have a cost within the suboptimality bound anymore. Instead, Optimistic Search uses Focal to find any solution and then by expanding nodes from Open tries to optimize the solution and prove it is w-optimal. Although there is no theoretical justification provided, this algorithm uses a different w for its Focal search and defines it as $w_f = 2 \cdot w - 1$.

Explicit Estimate Search (EES) (Thayer and Ruml 2011) is also a Focal list search algorithm that uses an additional heuristic named effort to go. Basically in this heuristic, we assume that the domain is unit cost meaning each action cost is one. In other words, as we apply this assumption, we are prioritizing the nodes which require fewer node expansions in order to reach the goal. Thus using this heuristic may result in finding the solution faster. Although it seems reasonable and can make EES efficient, in practice this algorithm is more challenging to be implemented as it uses different heuristics and multiple priority queues.

As the recent work (Chen and Sturtevant 2019) studied the issue of node re-expansions, it also suggested new Φ functions that can be used in search. In this paper, we will talk about the two Φ functions. The convex downward curve $\Phi_{XDP}(x, y) = \frac{1}{2w}[y + (2w-1)x + \sqrt{(y-x)^2 + 4wxy}]$, and the convex upward curve $\Phi_{XUP}(x, y) = \frac{1}{2w}(y + x + \sqrt{(y+x)^2 + 4w(w-1)x^2})$. As mentioned before, to use these functions, $f(n)$ has to be defined as $\Phi(h(n), g(n))$. The requirement of the Φ_{XDP} curve is for the path to be nearly optimal in the beginning of the search, while allowing for greater suboptimality as the search approaches the goal. Conversely, the Φ_{XUP} curve demands that the path be nearly optimal near the goal, with increasing tolerance for suboptimality as the distance from the goal increases. In contrast to the WA* algorithm, which allows for the same degree of suboptimality throughout the entire search path, the Φ_{XDP} and Φ_{XUP} curves can be used to guide the search towards a solution that meets specific suboptimality constraints.

3 Problem Definition

In suboptimal heuristic search problem, w is defined as the suboptimality bound on the returned solution meaning the path cost of that solution must be less than or equal to the cost of the optimal solution for the same problem. The cost of the solution is defined as $g(n)$ where n is the goal state. In suboptimal search, we will use a heuristic which both admissible and consistent. We call a heuristic function admissible if it never overestimates the cost from a node to the goal state and a consistent heuristic is defined as $h(a) \leq h(b) + d(a, b)$ where a and b are two arbitrary nodes and $d(a, b)$ is the optimal cost between them.

As mentioned before, the aim of this paper is to introduce a new framework named Improve Optimistic Search which is a suboptimal search algorithm. We will run IOS on both Sliding Tile Puzzle and Pancake Puzzle domains and generate w-optimal solution for each sample in these domains. In addition, we will demonstrate several studies on different attributes of this algorithm.

4 Improved Optimistic Search

Improved Optimistic Search is a search algorithm that operates within a bounded suboptimality framework, utilizing two distinct searches with different objectives. The first search is designed to quickly find a solution within a relatively small suboptimality bound by expanding nodes in the FOCAL list. Meanwhile, the second search is dedicated to verifying that the solution found by the first search meets the specified bound by expanding nodes in the OPEN list. This verification process ensures that the suboptimality of the solution is not exceeded.

Although it may seem more efficient to have only one list, by separating these two searches, IOS is able to find solutions quickly while ensuring that they meet the specified suboptimality constraints. To achieve this, these two searches use different priority functions to sort the node inside their priority queues. Open list is the standard A^* open list in which $\Phi(h(n), g(n)) = g(n) + h(n)$. On the other hand, different Φ functions can be used in Focal list that we will discuss more later in this chapter. All the Φ functions used in Focal use the same bound of $w_f = 2 \cdot w - 1$.

Algorithm 1 demonstrates the IOS algorithm. the general idea of this approach is easy to understand and implement. As the algorithm starts, it expands nodes from Focal and appends their children to it. It continues this procedure until it reaches the the goal state. Then, extracts the path from the start to the goal and updates the $c(I)$ which is the path cost.

Algorithm 1 Improved Optimistic Search $(start, goal, w)$

```
1: $Push\_Open(start)$
2: $Push\_Focal(start)$
3: $I \leftarrow empty\,[c(I) = \infty]$
4: **while** $\neg\ c(I)\ w\_optimal$ **do**
5:     **if** est. path length of best on Focal $< c(I)$ **then**
6:         Expand best from Focal
7:         **if** $best == goal$ **then**
8:             $I \leftarrow path(best)$
9:         **end if**
10:        Append successors to Focal
11:    **else**
12:        Expand best from Open
13:        **if** child of best was expanded on Focal **then**
14:            **if** new child cost < old child cost **then**
15:                *// Choose on of the following policies:*
16:                *(a) Update the cost of child on Focal*
17:                *(b) Re-open the child on Focal*
18:                **if** child $\in I$ **then**
19:                    *(c) Update cost of I // Solution-Update*
20:                **end if**
21:            **end if**
22:        **end if**
23:        Append successors to Open
24:    **end if**
25: **end while**
```

Once a solution is found, it starts expanding nodes from the Open until the w-optimality is proven.

To be more precise, while expanding nodes from Open, we may generate a node s which has been expanded in the first search using Focal. As the search in Focal is w-optimal and in Open we are expanding nodes based on the A^* priority function, we may find a shorter path to s using Open. Hence, we can update the previously found solution path, and change it to a path with a smaller cost. In other words, using Open, we can optimize the found solution. At each iteration of the main while loop, we only expand one node and we may apply one update to the solution. Hence, we only continue this procedure until the found solution cost is within the desired suboptimality range.

There exist different policies that can be used in Open in the mentioned scenario. In Section 4.1 we will describe the termination conditions in the main while loop and introduce what new condition can be used to improve the performance. Section 4.2 describes the different re-expansion policies and finally section 4.3, describes the possible Φ functions used in Focal.

4.1 Termination and Proving Bounds

Existing algorithms that ere mentioned in Section 2, like Optimistic Search and EES, use the best nodes f cost on Open to prove the suboptimality of the solution found. In particular, the best node on Open has the minimum f cost among all the nodes in Open and as the f cost never decreases in Open list, it can be used as a lower bound on the optimal solution cost. In other words, consider we find a solution by expanding the best node with the minimum f cost on Open. To prove this is the optimal solution, imagine the case in which we find another solution by expanding a node from Open later in the search. If the second solution has a fewer cost than the prior one, its f cost has to be less than the f cost of the first one since in the goal state f cost is equal to the solution cost. But we know that f cost in Open list using an admissible and consistent heuristic never decreases. So f cost of the second solution can not be less than the first one's and the first solution found is the optimal one. Thus, a solution found in Focal with a cost less than or equal to $w \cdot f_{min}$ is proven to be w-optimal. This explains the reason of using the condition $(c(I) < w \cdot f_{min})$ on the main while loop.

The typical Φ function used in WA* is $g(n) + wh(n)$. As in the goal state $h(n) = 0$, this priority function represents the estimated solution cost. On the other hand, the priority function $f'(n) = \frac{g(n)}{w} + h(n)$ is estimate of the optimal solution cost. The intuition behind this difference is that using the first priority function, we we may expand nodes with high g costs, as the h has more influence on sorting the nodes and expanding them, but this does not happen in the second priority function.

As we are using $w_f¿w$ in our Focal list along with WA* priority function, and we know that WA* does not need to re-expand states to achieve w-optimality, when the search expands a node with cost of f', it is proven that the optimal solution cost is greater than or equal to f' cost. Otherwise to reach to optimal solution, we would be forced to re-expand some states. Now recall that in suboptimal search, the objective is to find a solution that its cost is less than or equal to the w times the optimal cost. Thus, if we find a solution with cost less than or equal to $w \cdot f'$, as we now f' is also less than or equal to the optimal cost, we have achieved our objective. This implies that in IOS, the maximum f' cost of any state expanded from Focal can be used as an extra termination condition for the main while loop in the search. We define the notion of f'_{max} for this matter. To summarize, the second termination condition for IOS (also known as the improved termination condition) is when $c(I) \leq wf'_{max}$. It is worthy to mention that if WA* would always find solutions that were near the the upper-bound (exactly w-optimal), then it would not be useful to define this second termination condition. But, in practice the found solutions are much closer to the optimal solution than the upper-bound.

4.2 Re-expansion Policies

This section aims to study the effects of node re-openings and re-expansions during search, following the improved termination conditions discussed earlier. Node re-openings occur when a previously expanded node is placed back on a priority queue due to the discovery of a shorter path, while re-expansions occur when a node is expanded again after being re-opened. This section provides empirical evidence that node re-openings can have a substantial impact, either positive or negative, on search performance.

The example (a) in Figure 1 (Chen et al. 2019) demonstrates how good allowing the re-openings can be and it can saves large amount of unnecessary expansions. In this scenario, the search starts at the node S, the heuristic costs are mentioned using the red color and the cost of traversing be-

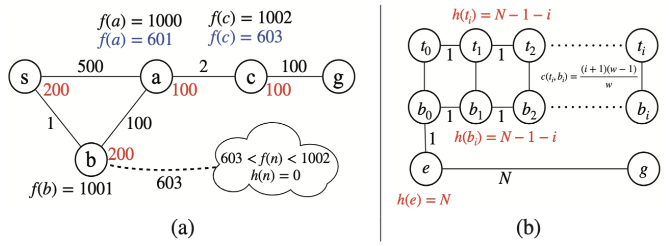


Figure 1: (a)Best and (b)Worst case of re-openings

tween any two nodes are mentioned on the line connecting them. The weight used in this example is 5 and WA* is used as the priority function for the search. As the search starts with the node S, both nodes a with a f cost of 1000 and b with a f cost of 1001 would be appended to the Open. Clearly a would be expanded and node c with a f cost of 1002 would be appended to the Open list. Now b would be expanded and we reach the critical state of this example. Following the scenario, as all the nodes in the cloud have the f cost of greater than 603 and less than 1002 (which is the f cost of node c), we have to expand them all before expanding the c. Then, c and finally the goal would be expanded.

The point is we can avoid all the unnecessary expansion in the cloud, by allowing node re-openings in this problem. Recalling the state after expanding b, we could re-open a as its new f cost is 601 and it is less than its previous cost. Since $601 < 604$, a and following that c and g would be re-expanded.

The second example in Figure 1, shows the potential disadvantage of allowing node re-openings. Although it is an arbitrary example, it could happen in real-world problems. In this scenario, search starts at t_0 and h cost of each state is mentioned with the red color. The cost of traversing between t_i and b_i is $c(t_i, b_i) = \frac{(i+1)(w-1)}{w}$. If we allow re-openings, using any arbitrary N and w, we would face expanding all the nodes from t_0 to t_i, following by expanding b_i. The general theme is to move from b_i to b_0 but what happens is that as we expand each b_k, we would need to re-open all the nodes back to b_i, and re-expand all of them from again. in general, the t nodes would be expanded one, but the bottom b nodes wold have $1+2+...+N = \frac{N(N+1}{2}$ cumulative expansions. Thus, we have $2N+2 = O(N)$ nodes in the graph and $\frac{N(N+1}{2}$ (b nodes)+N(t nodes)+2(two last nodes) = $O(N^2)$ expansions. This is identical to the worst-case scenario of A^*.

Summarizing what has been discussed here, in these two examples, we demonstrated how re-openings can affect the performance of the WA*. It both can help us to achieve large savings or to face $O(N^2)$ total expansions. Thus, it depends on the domain whether it is good to allow re-openings.

As re-expansions never happen using a consistent heuristic in Open, there only exist two scenarios in which we need to re-expand a state and both are in Focal. First, is when we expand a node in Focal and a shorter path to one of the state in Focal is found, and second one is when we expand a node in Open and we find a shorter path to a state in Focal. In both scenarios, there exist three policy that we can use. (1) Is the re-open policy in which we re-open the state in Focal and remove it from the Focal visited list. (2) Is the update policy in which we do not re-open the state but just update its g cost. And (3) is to update the solution cost ($c(I)$). Assume that we generate a node s in the Open and we observe that it has been visited before using Focal. In this case, if s was one of the nodes on the found solution path, all we have to do is to reduce the difference of its new g cost and old g cost from $c(I)$. As the search continues until $c(I) < w \cdot f_{min}$, it affects the search to avoid expanding extra nodes and terminate faster.

4.3 Alternate Priority Functions for Focal

As it may be advantageous for Improved Optimistic Search to minimize re-expansions, this study investigates IOS with priority functions for the Focal list that do not necessitate re-opening states when an expansion from Focal yields a shorter path to another state on Focal. This approach seeks to minimize the negative impact that re-expansions can have on the algorithm's performance. Therefore, we examine three options and all of them are independent of the weight used in Focal (we always the weight of $w_f = 2 \cdot w - 1$ in Focal priority functions). They are WA*, Φ_{XDP} and Φ_{XUP}.

The difference between Φ_{XDP} and Φ_{XUP} priority functions is where they allow the suboptimality. Despite WA*, in which the suboptimality is distributed uniformly along the search, in Φ_{XDP} and Φ_{XUP} priority functions the suboptimality changes in path from start to goal. As it is shown in Figure 2 (Chen et al. 2019), for Φ_{XDP} the slope increases as the search approaches toward the goal meaning that Φ_{XDP} is near-optimal when the search is close to the start state and increases its suboptimality in the way (we start the search at a state with a g cost of zero and ends it with a state with h cost of zero). On the other hand, the slop of the Φ_{XUP} decreases at it moves toward the goal meaning that it has its most suboptimality near the start and it is near-optimal when it is close to the goal state. These happens dues to the difference between the priority functions of them. The slop of WA* is the same all the way through the search as its priority function is $g(n)+wh(n)$ it the slop never changes.

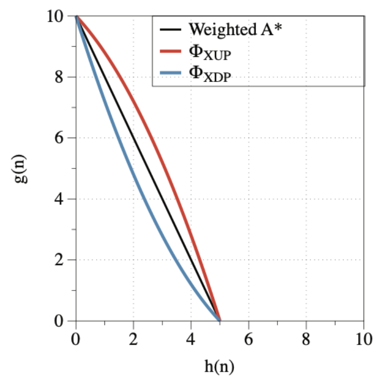
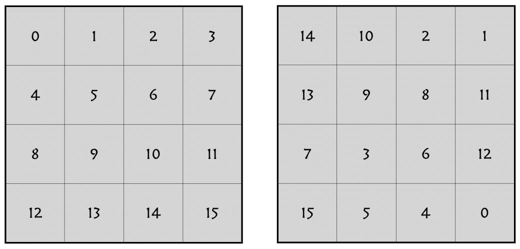


Figure 2: Sample isolines associated with WA*, Φ_{XDP} and Φ_{XUP}



(a) Goal State (b) Random Initial State

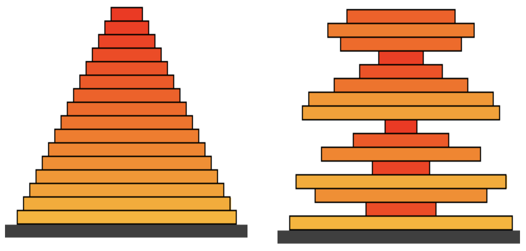
Figure 3: Sliding Tile Puzzle

5 Implementation and Experimental results

The experimental section of this paper aims to investigate the effectiveness of different pieces and strategies used in IOS. Hence, this section is broken into two smaller sections. The specification of the system used to run these experiments is an M1 Pro processor and 16 GB RAM. The used programming language is C++ with a clang compiler. To run the experiments, Sliding Tile Puzzle (STP) and Pancake Puzzle domains are used. The dataset used for the STP in the 100Korf samples and for the Pancake puzzle 50 random problems.

Each STP is a 4x4 board that includes 15 unordered tiles numbered from 1 to 15 in addition to one empty square indicated by the number 0. The goal is to find the sequence of actions that applying them in order, will reach us to the STP goal state shown in Figure 3. Each action is sliding one of the tiles to the empty square. Since only sliding is allowed, just the tiles which are adjacent to the empty square can be moved. In the heavy STP domain, the action cost of moving a tile is not one. Instead, it is equal to the value of the tile that is moved. Also in computing its heuristic, we should consider sum of the Manhattan distance of each tile to its original position times the value of that tile (Gilon, Felner, and Stern 2016).

Each Pancake Puzzle, includes 16 pancakes with different sizes stacked on top of each other. The objective to start from a random shuffled state, and reach the goal state in which all the pancakes are sorted in decreasing order of their sizes. Goal state and a random Initial state of this problem is demonstrated in Figure 4. In the heavy variant of the Pancake Puzzle, the cost of flipping pancakes $V[1] ... V[i]$ is $max(V[1], V[i])$ and i computing heuristic, we should generalize the GAP heuristic so we increase the h by $min(x, y)$ for each gap between pancakes x and y.



(a) Goal State (b) Random Initial State

Figure 4: Pancake Puzzle

5.1 Study of Improved Termination Condition

In this study, we observe the impact of the improved termination condition introduced in section 4.1. As it is illustrated in Tables 1, 2, and 3 as we decrease w used as the bound, the gain of applying the improved termination condition increases. This study is done by using the WA* as the priority function of the Focal list.

Bound	Using Cond.	not Using Cond.	Gain
1.25	295390	332628	11.19%
1.50	42436.9	50259.5	15.56%
2.00	11194.3	10770.8	3.78%
3.00	4557.85	4551.55	0.13%

Table 1: Average node expansions for IOS using and without using improved termination using WA* in STP

Bound	Using Cond.	not Using Cond.	Gain
1.25	277659	352899	21.32%
1.50	113660	144759	21.48%
2.00	48573.2	58585.5	17.09%
3.00	47104	47144.4	0.18%

Table 2: Average node expansions for IOS using and without using improved termination using WA* in Heavy STP

Bound	Using Cond.	not Using Cond.	Gain
1.25	637.63	756.34	15.69%
1.50	360.37	362.14	0.48%
2.00	350.41	350.41	0.0%
3.00	350.41	350.41	0.0%

Table 3: Average node expansions for IOS using with and without using improved termination using WA* in Pancake

5.2 Study of Alternate Priority Functions

In the last analysis of this research, the impact of best previous enhancements with various priority functions used for the Focal list is examined. As previously stated, only the priority functions Φ_{XDP}, WA*, and Φ_{XUP} are taken into account as they avoid node re-expansions. We provided the results of this comparison across all the two domains in Tables 4 and 5.

As it is shown, IOS is able to solve the all the problems using these priority functions except the Heavy Pancake domain where some of the algorithms were not able to solve the problems with small weights. IOS with the Φ_{XDP} has the best performance across almost all the domains and weights. In the recent paper of (Chen et al. 2019) the results of comparison of these three priority functions in the Heavy STP is a little different. In the experimental results provided there, WA* performed better than Φ_{XDP} in smaller weight. On the other hand in our experiment, Φ_{XDP} also performed better in smaller weights. In fact, we also expected Φ_{XDP} to perform better in smaller weights according to the other recent work(Chen and Sturtevant 2019).

Bound	Φ_{XDP}	WA*(w_f)	Φ_{XUP}
1.25	**189476**	295390	462248
1.50	**24060.3**	42436.9	65235.3
2.00	**6738.44**	10770.8	14807.8
3.00	**3888.5**	4551.55	6756.26

Table 4: Average node expansions for IOS using different Φ functions for Focal in STP

Bound	Φ_{XDP}	WA*(w_f)	Φ_{XUP}
1.25	**184336**	277659	549536
1.50	**75022.8**	113660	178404
2.00	**39704.1**	48573.2	69864.9
3.00	44967.6	47104	**39988.4**

Table 5: Average node expansions for IOS using different Φ functions for Focal in Heavy STP

References

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

Chen, J.; and Sturtevant, N. R. 2019. Conditions for avoiding node re-expansions in bounded suboptimal search. *puzzle*, 40: 39–753.

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.

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.

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

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

Bound	Φ_{XDP}	WA*(w_f)	Φ_{XUP}
1.25	177.84	99	67.82
1.50	61.94	42.6	41.52
2.00	43.18	40.68	40.68
3.00	40.04	40.68	40.68

Table 6: Average node expansions for IOS using different Φ functions for Focal in Pancake