Type-WA*: Using Exploration in Bounded Suboptimal Planning

Eldan Cohen[1,2] , **Richard Valenzano**[3] and **Sheila A. McIlraith**[1,2,4]
[1]University of Toronto, Toronto, Canada
[2]Vector Institute, Toronto, Canada
[3]Ryerson University, Toronto, Canada
[4]Schwartz Reisman Institute for Technology and Society, Toronto, Canada
ecohen@mie.utoronto.ca, rick.valenzano@ryerson.ca, sheila@cs.toronto.edu

Abstract

Previous work on satisficing planning using greedy best-first search (GBFS) has shown that non-greedy, randomized exploration can help escape uninformative heuristic regions and solve hard problems faster. Despite their success when used with GBFS, such exploration techniques cannot be directly applied to bounded suboptimal algorithms like Weighted A* (WA*) without losing the solution-quality guarantees. In this work, we present Type-WA*, a novel bounded suboptimal planning algorithm that augments WA* with type-based exploration while still satisfying WA*'s theoretical solution-quality guarantee. Our empirical analysis shows that Type-WA* significantly increases the number of solved problems, when used in conjunction with each of three popular heuristics. Our analysis also provides insight into the runtime vs. solution cost trade-off.

1 Introduction

Optimal planning algorithms, such as A*, are guaranteed to return optimal solutions, but they do so at the expense of high runtime and memory requirements. Most practical applications dispense with optimality guarantees, often favoring *bounded suboptimal* algorithms which generally have far more modest runtime and memory requirements, while also providing quantifiable quality guarantees.

The most popular bounded suboptimal planning algorithm is *Weighted A* (WA*)* [Pohl, 1970]. WA* and its variants have been used in a wide range of applications including BDD minimization [Ebendt and Drechsler, 2009] and path planning [Bono *et al.*, 2019; Zeng *et al.*, 2015]. They have also been extensively used in robotics applications [e.g., Vernaza *et al.*,2009; Cohen *et al.*, 2014]. WA* is also a key component in anytime algorithms such as Anytime Restarting Weighted A* [Richter *et al.*, 2010] and Anytime Repairing A* [Likhachev *et al.*, 2003], and is often used in domain-independent planning systems such as LAMA [Richter and Westphal, 2010] and FD stone soup [Helmert *et al.*, 2011].

Despite its broad adoption, WA*'s performance often degrades in uninformative heuristic regions (UHRs), where the heuristic does not provide useful guidance regarding how to make progress [Wilt and Ruml, 2012]. In *greedy best-first search (GBFS)*, a popular algorithm for satisficing planning that provides no solution-quality guarantees, a recent line of work has considered using non-greedy, randomized *exploration* to overcome UHRs. For example, Type-based exploration in GBFS [Xie *et al.*, 2014], that utilizes type systems to explore the state space, was adopted by state-of-the-art satisficing planners (e.g., FD stone soup 2018 [Seipp and Röger, 2018]). However, such techniques cannot be directly applied to WA* without losing the solution-quality guarantees.

In this work, we extend type-based exploration to support bounded suboptimal planning by restricting the exploration to the focal list. We present *Type-WA**, a novel planning algorithm that augments WA* with type-based focal exploration, while still satisfying WA*'s theoretical solution-quality guarantee. To the best of our knowledge, this is the first algorithm that uses non-greedy, randomized exploration while still being bounded suboptimal. Our contributions are as follows:

1. We introduce Type-WA* and prove it is w-admissible.
2. We present an extensive empirical analysis of Type-WA* across domains, heuristic functions, and weights. This analysis shows that Type-WA* significantly outperforms WA* in terms of runtime and the number of instances solved in given time and memory limits.
3. We discuss how Type-WA* can avoid pathological deficiencies exhibited by WA*. In particular, we identify that Type-WA* can overcome the problem of performance degradation for larger weights [Wilt and Ruml, 2012] exhibited by WA* in some domains.
4. We show how the level of exploration in Type-WA* can be controlled and we empirically analyze the impact that doing so has on solution quality.

2 Background

A *search problem* is a tuple $\langle G(V,E), s_i, T\rangle$. G is a finite directed graph called the *state space*, with *states* or *vertices* V and *edges* E. $s_i \in V$ is the *initial state*, and $T \subseteq V$ is the set of *goal* states. We use (s,s') to denote an edge from s to s', and $successors(s)$ to denote the set of states $\{s' \mid (s,s') \in E\}$. The objective is to find a *solution path*, a sequence of states $\langle s_0, ..., s_{n-1}\rangle$ such that $s_0 = s_i$, $s_{n-1} \in T$ and $s_j \in successors(s_{j-1})\ \forall j \in [1..n{-}1]$.

A *cost* function $c : E \rightarrow \mathbb{R}^{\geq 0}$ assigns a cost to each edge. The cost of a path is the sum of the edges along that path. A *bounded suboptimal problem* is then given by $\langle G(V, E), s_i, T, c, w \rangle$, where $w \geq 1$. The objective is to find a solution path with a cost of no more than wC^*, where C^* is the cost of an *optimal* solution path (i.e., one with lowest cost)[1]. Such solutions are said to be w-admissible. If every solution returned by an algorithm satisfies such a requirement for a given w, that algorithm is also said to be *w-admissible*.

A *node* n is given by a state s and a path from s_i to s. This path is often stored implicitly using a parent pointer $pred(n)$, which points to the previous node along the path. The g-cost of node n, denoted $g(n)$, is the cost of this path. We also use $g^*(n)$ for the cost of the optimal path from s_i to s, and $h^*(n)$ for the cost of the optimal path from s to a state in T.

2.1 Heuristic Search Algorithms

Many classical algorithms search for a solution by iteratively generating partial paths of nodes until one is found that reaches a goal. We assume the reader's familiarity with the use of *open* and *closed* lists of nodes, whereby nodes on the open list are selected, *expanded*, and moved to the closed list[2]. These algorithms differ in the policy they use to select nodes from the open list on every iteration. For example, *best-first search (BFS)* algorithms use some evaluation function Φ from the set of nodes to $\mathbb{R}$ to define this policy. In particular, on every iteration, a BFS algorithm selects the node from the open list with the lowest value according to Φ.

Many BFS algorithms use a *heuristic function* h to define this evaluation function. For example, the *A** algorithm uses $\Phi(n) = f(n) = g(n) + h(n)$. We refer to $f(n)$ as the f-cost of a node. In contrast, the *Weighted A* (WA*)* algorithm uses the evaluation function $\Phi(n) = f_w(n) = g(n) + wh(n)$, where $w \geq 1$ is a user-defined parameter called a *weight* [Pohl, 1970]. Notably, WA* is w-admissible if the heuristic is *admissible*, i.e., if $h(n) \leq h^*(n)$ for any node n.

Focal search algorithms [Pearl and Kim, 1982] are an alternative to BFS algorithms. For the node selection step, these algorithms use a heuristic h to first identify $f_{\min}$, which is the minimum value of f-cost of all nodes in the open list. Next, these algorithms construct the *focal* list, which is a subset of the open list defined as follows:

$$FOCAL = \{n \in OPEN \mid f(n) \leq w \cdot f_{\min}\}. \quad (1)$$

Finally, a single node is selected for expansion from FOCAL, using some other policy. The first focal search algorithm, A^*_ϵ [Pearl and Kim, 1982], selects nodes in FOCAL according to some secondary, potentially inadmissible, heuristic. Other focal search algorithms, like EES [Thayer and Ruml, 2011], use a more sophisticated policy for selecting nodes.

2.2 GBFS and Random Exploration

Greedy Best-First Search (GBFS) is a BFS algorithm that uses $\Phi(n) = h(n)$. Thus, GBFS always expands the node with the lowest h-cost in the open list. Doing so generally speeds up the search for solutions, though there are no general guarantees on the quality of solutions found.

Due to the way it exploits the heuristic function, GBFS can be easily led astray by misleading heuristics. For example, the search can often become "stuck" in uninformative heuristic regions (UHRs) of the state space, in which the heuristic does not provide guidance regarding how to make progress. To alleviate this effect, a variety of methods that use *random exploration* [Valenzano *et al.*, 2014; Xie *et al.*, 2014; Imai and Kishimoto, 2011; Asai and Fukunaga, 2017] have been proposed. These methods use some form of random sampling when selecting nodes for expansion, to encourage the algorithm to occasionally go against the advice of the heuristic. *Type-GBFS* [Xie *et al.*, 2014], which is our focus, bases its approach for random sampling on a *type system*:

Definition 1 (Type System [Lelis *et al.*, 2013])**.** Let N be the set of nodes in a search space. $T = \{t_1, \ldots, t_n\}$ is a type system for N if T is a disjoint partitioning of N. For every node $n \in N$, $T(n)$ denotes the unique $t \in T$ for n.

For example, the (h, g) type system introduced in the original Type-GBFS paper will put two nodes n and n' in the same partition if and only if $h(n) = h(n')$ and $g(n) = g(n')$.

When selecting a node for expansion, Type-GBFS alternates between selecting the node with the lowest heuristic value (i.e., the same node as selected by GBFS), and selecting a node by random sampling as follows. First, the nodes in the open list are grouped by their type. Next, a type t_i is selected uniformly at random from those that are non-empty after this grouping. Finally, a node is selected uniformly from amongst those in the open list that are of type t_i.

The intuition behind Type-GBFS is to encourage the exploratory node selections to better cover the state space. For example, when most of the open list is dominated by nodes from one region of the state space (i.e., a "bad" subtree or UHR), exploration methods which uniformly sample nodes from the open list (e.g., ϵ-greedy node selection [Valenzano *et al.*, 2014]) are likely to select nodes from that same region. In contrast, if the type system groups many of the states in that dominant region together, there is a better chance that Type-GBFS will sample nodes outside of the UHR [Xie *et al.*, 2014].

There are a variety of other methods for introducing random exploration to GBFS (cf. [Imai and Kishimoto, 2011; Asai and Fukunaga, 2017]). Though we omit a full description of these methods, we note that they can be used in the bounded suboptimal setting using similar approaches to those described in the next section.

3 WA* with Type-Based Focal Exploration

To motivate our approach, we start by demonstrating that UHRs are also a problem for WA*. Consider an instance of NoMystery with 14 packages and 14 locations, which is solved by WA* with $w = 3$ using the admissible landmark count heuristic [Karpas and Domshlak, 2009] in approximately 36 minutes. Figure 1 shows the distribution of f_w values in the open list after 50K, 100K, 500K, and 1M expansions (note the log-scaled y-axis). The dashed line marks

[1]Other forms of solution-quality requirements can also be defined [Valenzano *et al.*, 2013]. While our approach is applicable for many of them, we focus on w-admissibility for the sake of clarity.

[2]In this work, we assume that nodes on the closed list are *re-opened* whenever a shorter path to them is found.

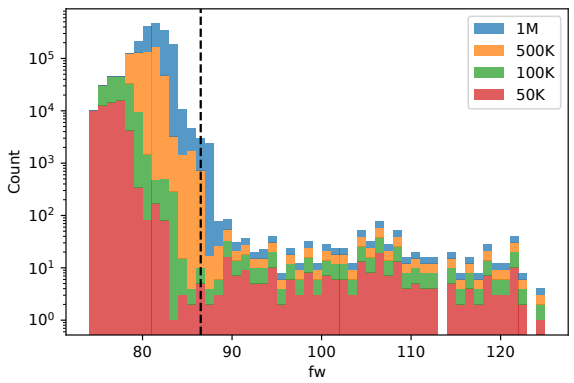


Figure 1: Distribution of f_w values in the WA* open list at different time points. The problem instance is from the NoMystery domain.

the f_w value of the next solution node according to the solution that is returned, eventually, by WA*. We see that the overwhelming majority of nodes in the open list have lower f_w value compared to the solution node. This distribution indicates a large UHR that will have to be exhausted before the solution node is expanded. We hypothesize that in many domains, such UHRs account for significant difficulty.

As discussed above, several works have used exploration as a way to alleviate the effect of large UHRs. Although these methods cannot be applied directly to WA* while guaranteeing w-admissibility, in the remainder of the section we identify an approach for augmenting WA* with exploration while still maintaining bounded suboptimality.

3.1 The Type-WA* Algorithm

Motivated by the above analysis, we propose *Type-WA** (or *TWA** for short), a novel bounded suboptimal planning algorithm that augments WA* with type-based exploration in the focal list. Algorithm 1 shows a pseudocode for the proposed approach. It alternates between expanding nodes based on f_w (as in WA*) and expanding a random node from the focal list using type-based exploration. By constraining the type-based exploration to the focal, Type-WA* is guaranteed to find solutions that are w-admissible, as shown in Theorem 1.

Theorem 1 (Bounded Suboptimality)**.** *If Type-WA* returns a solution for a weight w and admissible heuristic h, then the solution is w-admissible.*

Proof Sketch. Type-WA* always expands a node n that is either (a) the open node with the lowest f_w value (i.e., the weighted A* selection criterion) or; (b) a node from the focal list.

If an algorithm only expands nodes from the focal list computed for admissible heuristic h and weight w, it is guaranteed to return a w-admissible solution. Since Ebendt and Drechsler [2009] showed that expanding the lowest f_w value (as in WA*) is guaranteed to be in FOCAL, Type-WA* only expands nodes that are in FOCAL. □

We note that any of the other methods for inducing exploration can also be used in a bounded suboptimal search, if that exploration is similarly restricted to the focal list. We leave a full examination of such methods as future work.

Algorithm 1 The Type-WA* Algorithm

Input: init node n_I, heuristic h, weight w, Type system T

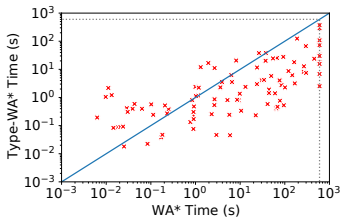
```
1:  function TYPE-WA*(n_I, h, w, T)
2:    g(n_I) = 0, pred(n_I) = NONE
3:    OPEN ← {n_I}, CLOSED ← {}, step ← 1
4:    while OPEN ≠ ∅ do
5:      if step is odd then                          ▷ Weighted A* step
6:        n ← arg min_{n∈OPEN} f_w(n) = g(n) + wh(n)
7:      else                          ▷ Type-based focal exploration step
8:        FOCAL = {n | n ∈ OPEN, f(n) ≤ w · f_min}
9:        t ← randomly-selected, non-empty type from T
10:       n ← randomly selected FOCAL node in type t
11:     if n is a goal then
12:       return path to n
13:     for n_c ∈ successors(n) do
14:       if n_c ∉ OPEN ∪ CLOSED then                ▷ New state
15:         g(n_c) = g(n) + c(n, n_c)
16:         pred(n_c) ← n
17:         OPEN = OPEN ∪ {n_c}
18:       else if g(n) + c(n, n_c) < g(n_c) then
19:                                          ▷ Found lower-cost path
20:         g(n_c) = g(n) + c(n, n_c)
21:         pred(n_c) ← n
22:         if n_c ∈ CLOSED then                     ▷ Reopen node
23:           CLOSED ← CLOSED − {n_c}
24:           OPEN ← OPEN ∪ {n_c}
25:     CLOSED ← CLOSED ∪ {n}
26:     step = step + 1
27:   return ∅
```

Implementation Details

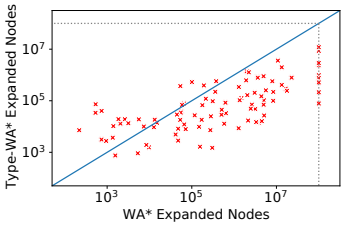
In practice, focal search algorithms maintain multiple data structures to allow for the incremental tracking of f_{min} and the contents of the focal list. For Type-WA*, we also need to maintain a grouping of nodes by types. We use the following approach, which is suitable when the type system satisfies the property that for any two nodes n and n' for which $f(n) \neq f(n')$, n and n' are in different types. We note that this is satisfied by the (h, g) type system used below, and other common type systems like (f).

First, we maintain an open list ordered by f_w, to allow easy access for the WA*-like expansions. For the focal list, we store a bucket of nodes for each type. Since all nodes in the same bucket have the same f-cost, we can sort the buckets by f-cost. Doing so allows for an easy calculation of $f_{\min}$, $wf_{\min}$, and thereby the set of buckets corresponding to nodes in the focal list. For an exploration expansion, we then simply randomly select one of the buckets in this set, and then randomly return a node in the selected bucket.

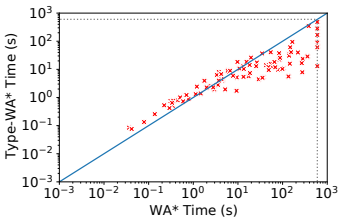
The open list ordered by f_w and the type buckets are not synchronized, and some nodes in them may already be closed. We simply ignore these nodes if they are selected for expansion. But we do need to maintain $f_{\min}$ accurately, since underestimating its true value can restrict the focal list and consequently the amount of exploration. Therefore, before selecting a node for an exploratory expansion, we remove nodes from the lowest f-cost bucket until we find one that is open.



(a) $w = 2$: Time (seconds) using h^{LM}.

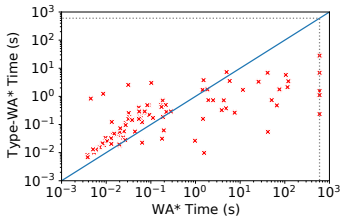


(b) $w = 2$: Expanded nodes using h^{LM}.

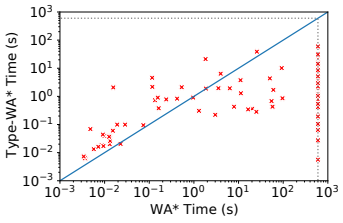


(c) $w = 1.5$: Time (seconds) using h^{LC}.

Figure 2: Results for 100 random Blocksworld problem instances (note the log-scaled axes; the dotted lines represent timeout).



(a) $w = 2$: Time (seconds) using h^{MS}.



(b) $w = 3$: Time (seconds) using h^{MS}.

	h^{MS}		h^{LM}	
w	WA*	TWA*	WA*	TWA*
1.5	**100**	**100.0**	**100**	**100.0**
2	93	**100.0**	94	**100.0**
3	49	**100.0**	76	**100.0**
5	21	**100.0**	64	**100.0**

(c) # of solved instances (out of 100).

Figure 3: Results for 100 random NoMystery problem instances (note the log-scaled axes; the dotted lines represent timeout).

4 Empirical Analysis

In this section, we present an empirical analysis of Type-WA*. We start by analyzing its performance on two well-known benchmark domains: Blocksworld and NoMystery. Then, we present an extensive evaluation on problem instances from the International Planning Competition (IPC).

We tested with three different admissible heuristics: the *merge-and-shrink heuristic* (h^{MS}) [Helmert *et al.*, 2014], the admissible variant of the *landmark count heuristic* (h^{LM}) [Karpas and Domshlak, 2009], and the *landmark cut heuristic* (h^{LC}) [Helmert and Domshlak, 2009]. In all experiments we use a time limit of 10 minutes and a memory limit of 4GB. Experiments are conducted with the type system (h, g). Since Type-WA* is a stochastic algorithm, the reported coverage (i.e., the number of instances solved in the time and memory limits) for Type-WA* is *averaged over 5 runs*.

We implemented Type-WA* in the Fast Downward planner [Helmert, 2006], and validated plans using VAL [Howey *et al.*, 2004]. To maintain integer f values, we round down the weighted heuristic value in experiments that use fractional w, $f_w(n) = g(n) + \lfloor w \cdot h(n) \rfloor$. We run experiments on an AMD Ryzen Threadripper 2990WX.

4.1 Performance on Blocksworld Instances

We consider a set of 100 random instances of the 4-operator Blocksworld problem with 15 blocks. Figure 2a (Figure 2b) compares the search time (expanded nodes) of WA* and Type-WA* for $w = 2$ using h^{LM}. Points under the diagonal correspond to problems that were solved faster using Type-WA* and vice versa. For problems that were relatively easy for WA*, there is no significant difference with and without exploration. However, for problems that were relatively hard for WA* (e.g., problems that took more than 10 seconds), Type-WA* significantly outperforms WA* and manages to reduce the solution time and number of node expansions by up to several order of magnitudes. Specifically, while WA* only solved 87 instances in the time limit, Type-WA* solved an average of 99.8 instances. In experiments with a lower weight of 1.5, WA* solved only 74 instances while Type-WA* solved 92 instances. In experiments with a higher weight, $w = 3$, WA* solved 98 instances while Type-WA* solved all 100 instances. While Figures 2a and 2b are based on a single run of Type-WA*, repeated runs yielded similar trends. The reported coverage is averaged over five runs.

h^{LC} was found to be more effective for this problem, so we experimented with smaller weights: $w \in \{1.3, 1.5, 2\}$. Figure 2c compares the solution time for $w = 1.5$ and shows a similar pattern: the harder instances are solved faster with Type-WA*, including 7 instances that were not solved by WA*. Detailed results for h^{LM} and h^{LC} appear in Appendix A[3]. We note that results with h^{MS} are omitted, since all methods performed poorly when using this heuristic.

4.2 Performance on NoMystery Instances

Next, we consider a set of 100 random instances of NoMystery with 14 packages and 14 locations. Figure 3a compares WA* and Type-WA* for $w = 2$ using h^{MS}. Similar to Blocksworld, Type-WA* solves 7 problems WA* could not, and solves the hardest instances significantly faster.

Typically, increasing w tends to make problems easier for WA* at the cost of a weaker suboptimality bound. However, in this domain, we find that increasing w actually makes the problems harder for WA*. Figure 3b shows the results for the

[3]All appendices appear in Cohen *et al.* [2021].

Domain	h^{MS}		h^{LM}		h^{LC}	
	WA*	TWA*	WA*	TWA*	WA*	TWA*
barman (14)	3	**6.0**	3	**6.0**	0	**3.0**
cavediving (20)	7	7.0	7	7.8	3	3.0
childsnack (20)	0	**4.4**	0	**3.8**	0	**5.0**
citycar (20)	15	**18.2**	13	**19.6**	0	0.0
floortile (20)	6	5.2	2	2.0	12	**14.6**
ged (20)	20	20.0	15	**17.4**	19	19.8
hiking (20)	20	20.0	18	**19.6**	13	**14.2**
maintenance (5)	5	5.0	5	5.0	5	5.0
openstacks (20)	**3**	2.0	**3**	1.0	**3**	2.0
parking (20)	13	12.4	9	9.0	16	15.6
tetris (17)	7	7.0	12	**17.0**	8	8.4
tidybot (20)	7	**15.8**	17	**20.0**	**16**	15.0
transport (20)	7	**13.8**	**9**	8.0	8	8.4
visitall (20)	6	**9.0**	20	20.0	13	12.8
agricola (20)	0	**20.0**	0	**20.0**	0	**2.6**
caldera (20)	10	**16.6**	16	**20.0**	0	0.0
data-net. (20)	11	**18.0**	13	**18.0**	13	**17.6**
nurikabe (20)	11	**14.0**	16	**17.8**	0	0.0
organic. (20)	7	7.0	7	7.0	7	7.0
petri-net. (20)	**4**	3.0	2	2.0	12	12.0
settlers (20)	8	**11.8**	13	13.0	0	0.0
snake (20)	11	**14.8**	14	**15.8**	7	**9.2**
spider (20)	11	**12.0**	**19**	17.2	12	12.4
termes (20)	**15**	12.0	12	**14.2**	8	**12.0**
Total (456)	207	**275.0**	245	**301.2**	175	**199.6**

Table 1: Number of solved instances in IPC domains for $w = 3$.

higher weight $w = 3$. Now, we find 51 instances that could not be solved by WA* in the time and memory limit. Interestingly, the increase in weight does not have a negative impact on Type-WA*'s coverage, as it still solves all 100 instances. Figure 3c provides a detailed report on the number of problems solved by WA* and Type-WA* for different weights and heuristic functions (omitting h^{LC} which performed poorly on this domain). While increasing the weight seems to have a very negative impact on WA*, it does not similarly affect Type-WA*, which solves all 100 instances in all settings.

Wilt and Ruml [2012] analyzed WA* in domain-specific heuristic search and showed that WA* performance can deteriorate when increasing w. This is because increasing w makes the search increasingly greedy (i.e., increasingly based on the heuristic h). As such, if h suffers from large UHRs, increasing w can lead to degraded performance [Wilt and Ruml, 2012]. While previous work [Valenzano *et al.*, 2014; Xie *et al.*, 2014; Cohen and Beck, 2018] has shown that exploration can alleviate this issue when used with GBFS, our work suggests that by augmenting WA* with exploration in the focal list, we can successfully escape large UHRs while still maintaining w-admissibility.

4.3 Performance on IPC domains

We now evaluate our approach using problem instances from the International Planning Competition (IPC). We consider all the instances from the optimal track in IPC'18 and IPC'14 and compare the performance of WA* and Type-WA*. Table 1 shows the coverage for $w = 3$ when using the three heuristics: h_{MS}, h_{LM}, and h_{LC}. Recall that the coverage

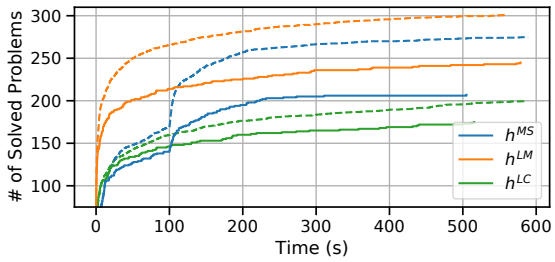


Figure 4: Coverage over time for Type-WA* (dashed lines) and WA* (solid line) on IPC problems.

for Type-WA* is averaged over five runs. We can see that, in total, Type-WA* achieves significantly higher coverage compared to WA* for each of the heuristics. Furthermore, there are some domains in which Type-WA* particularly outperforms WA* (e.g., agricola and tidybot for h^{MS}).

Figure 4 compares the coverage of WA* (solid line) and Type-WA* (dashed line) over time. The figure shows that in terms of coverage, Type-WA* dominates WA* at all time points with all three heuristics. We note that the performance pattern exhibited by h^{MS} is due to the typically longer heuristic computation time that we limit to 100 seconds.

Results for Different w

We analyzed the coverage for h^{MS} with $w \in \{2, 3, 10\}$. Type-WA* significantly improved the coverage for all weights and solved, on average, 60.2 more instances for $w = 2$, 68 more instances for $w = 3$, and 70.8 more instances for $w = 10$. A per-domain analysis appears in Appendix C.

Solution Cost and Exploration

To evaluate the impact of exploration on actual solution cost, we normalized the cost of each solution found using the lowest cost solution found by any of the configurations (i.e., C/C^{best}, where C is the cost of the obtained solution and C^{best} is the lowest cost solution found). Table 2 shows the average normalized cost for WA* and Type-WA* with h^{MS} for $w \in \{2, 3, 10\}$. Since the normalized cost is averaged only over solved problems, we also report the number of solved problems. The results are aggregated over all domains, however per-domain results appear in Appendix D.

The table shows that while Type-WA* solved a significantly higher number of problem instances, the solutions tend to have higher cost (though the suboptimality bound is still satisfied). These results suggest that Type-WA* makes use of the allowed suboptimality in order to solve more problems in the time limit. To further test this hypothesis, we ran Type-WA* with $w = 10$, while using a second lower weight,

	$w = 2$		$w = 3$		$w = 10$	
	WA*	TWA*	WA*	TWA*	WA*	TWA*
Cost	1.01	1.12	1.03	1.24	1.10	1.68
# Solved	196	256.2	207	275.0	227	297.8

Table 2: Normalized cost and # of solved instances on IPC domains.

	$w = 10$		
	WA*	TWA*	R-TWA*
Cost	1.10	1.68	1.14
# Solved	227	297.8	264.8

Table 3: Normalized cost and # of solved instances for IPC domains.

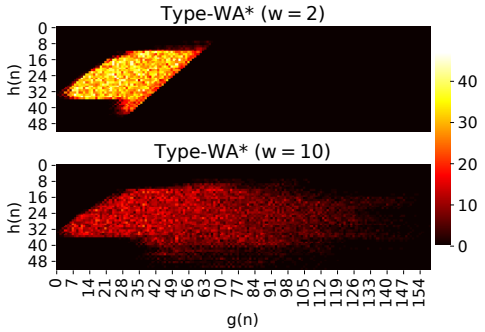


Figure 5: # of expanded nodes per type during exploration steps.

$\bar{w} = 2$, to construct the focal list.[4] The results are reported in Table 3, where, R-TWA* denotes Type-WA* with an open list ordered by $f(s) = g(s) + 10 \cdot h(s)$ and a restricted focal list that only contains nodes with $f(s) \leq 2 \cdot f_{min}$. We can see that restricting the exploration leads to a lower cost compared to standard Type-WA*, though fewer problems are solved. Note that R-TWA* still solved significantly more problems than WA*, while only incurring a minor increase in cost.

To account for this effect, we analyzed the nodes expanded by Type-WA* for $w = 2$ vs. $w = 10$, on a IPC'14 floortile instance when using h^{MS}. Figure 5 shows heatmaps of the number of nodes expanded by just the exploration steps for each (h, g) type during the first 50,000 expansions. We see that increasing w leads to a larger variation in explored types. This helps explain the gain in coverage as exploring more types may overcome UHRs and help find solutions faster. It also explains the decrease in solution quality, since the additional types being explored often have higher g-cost and thus lead to lower quality solutions. Note that R-TWA* with $w = 10$ and $\bar{w} = 2$ exhibits similar exploration patterns to Type-WA* with $w = 2$, which accounts for the lower coverage and higher solution quality.

Results for Pure Type-based Focal Exploration

We also analyzed the performance of a search that uses only type-based focal exploration and we report the results in Appendix E. We find that pure type-based focal exploration performs well and even outperforms WA* in some settings. Still, Type-WA* significantly outperforms both WA* and pure type-based focal exploration.

5 Discussion and Future Work

These experiments demonstrate that WA* can struggle due to UHRs, and in some IPC domains like NoMystery, we see the previously reported behaviour where a higher weight leads to worse performance [Wilt and Ruml, 2012]. We note that this phenomenon may also negatively affect anytime algorithms that use WA* with decreasing weights (e.g., RWA* [Richter *et al.*, 2010] and the LAMA planner [Richter and Westphal, 2010]). Our analysis also shows that Type-WA* better handles UHRs while still satisfying suboptimality bounds, and did not lead to significant degradation in performance due to higher weights. This suggests that Type-WA* may be a more robust choice in the context of such anytime algorithms. A detailed investigation of the empirical impact of using exploration in anytime planning is a direction for future work.

Our results suggest that Type-WA* appears to be exploiting the suboptimality bound to find solutions faster. We also found that restricting the focal list used for exploration can effectively trade-off solution quality with coverage. It is therefore interesting to investigate methods that dynamically adapt the level of exploration through the size of the focal list to better manage this trade-off. For example, we can try to extend the bound over time, thereby seeking higher quality solutions quickly if possible, before exploring nodes that are likely to lead to lower quality solutions if progress is not made.

Our approach for incorporating type-based exploration into WA* through the focal list can easily be extended to additional exploration techniques and other bounded suboptimal search algorithms. Further investigating this idea when using (potentially inadmissible) heuristics to sort the focal list may be beneficial, since the search is likely to be sensitive to UHRs induced by these heuristics.

Currently, Type-WA* supports any admissible heuristic including inconsistent heuristics, e.g., h^{LC}. Previous work on bounded suboptimal planning with consistent heuristics showed that the number of re-expansions can be bounded [Narayanan *et al.*, 2015], or even eliminated [Likhachev *et al.*, 2003; Chen and Sturtevant, 2019], without violating w-admissibility. Investigating whether we can adapt Type-WA* to bound the number of re-expansions when used with a consistent heuristic is an interesting direction for future work.

6 Conclusion

In this work, we presented the first bounded suboptimal planning algorithm that incorporates random exploration in the search. Our approach, Type-WA*, augments WA* with type-based exploration in the focal list, yielding solutions that satisfy the theoretical bound of WA*. We perform extensive empirical analysis across different domains, heuristics, and weights. We show that Type-WA* achieves significantly higher coverage than WA* and is less sensitive to UHRs. We provide insight into the impact of exploration and the runtime vs. solution cost trade-off. Our work demonstrates the potential of using exploration in bounded suboptimal planning and identifies several promising directions for future work.

Acknowledgements

We thank the anonymous reviewers for their valuable feedback. We gratefully acknowledge funding from NSERC, the CIFAR AI Chairs program (Vector Institute), and from Microsoft Research.

[4] Type-WA* is $\mathcal{W}$-admissible where $\mathcal{W} = \max(w, \bar{w})$.

References

[Asai and Fukunaga, 2017] Masataro Asai and Alex Fukunaga. Exploration among and within plateaus in greedy best-first search. In *ICAPS*, pages 11–19, 2017.

[Bono *et al.*, 2019] Massimo Bono, Alfonso Emilio Gerevini, Daniel Damir Harabor, and Peter J Stuckey. Path planning with CPD heuristics. In *IJCAI*, pages 1199–1205, 2019.

[Chen and Sturtevant, 2019] Jingwei Chen and Nathan R. Sturtevant. Conditions for avoiding node re-expansions in bounded suboptimal search. In *IJCAI*, pages 1220–1226, 2019.

[Cohen and Beck, 2018] Eldan Cohen and J Christopher Beck. Local minima, heavy tails, and search effort for GBFS. In *IJCAI*, pages 4708–4714, 2018.

[Cohen *et al.*, 2014] Benjamin Cohen, Mike Phillips, and Maxim Likhachev. Planning single-arm manipulations with n-arm robots. In *RSS*, July 2014.

[Cohen *et al.*, 2021] Eldan Cohen, Richard Valenzano, and Sheila A. McIlraith. Type-WA*: Exploration in bounded suboptimal planning (IJCAI-2021 supplementary material). Technical Report CSRG-638, Department of Computer Science, University of Toronto, May 2021.

[Ebendt and Drechsler, 2009] Rüdiger Ebendt and Rolf Drechsler. Weighted A* search–unifying view and application. *Artificial Intelligence*, 173(14):1310–1342, 2009.

[Helmert and Domshlak, 2009] Malte Helmert and Carmel Domshlak. Landmarks, critical paths and abstractions: What's the difference anyway? In *ICAPS*, pages 162–169, 2009.

[Helmert *et al.*, 2011] Malte Helmert, Gabriele Röger, and Erez Karpas. Fast downward stone soup: A baseline for building planner portfolios. In *ICAPS 2011 Workshop on Planning and Learning*, pages 28–35, 2011.

[Helmert *et al.*, 2014] Malte Helmert, Patrik Haslum, Jörg Hoffmann, and Raz Nissim. Merge-and-shrink abstraction: A method for generating lower bounds in factored state spaces. *JACM*, 61(3):1–63, 2014.

[Helmert, 2006] Malte Helmert. The fast downward planning system. *JAIR*, 26:191–246, 2006.

[Howey *et al.*, 2004] Richard Howey, Derek Long, and Maria Fox. VAL: Automatic plan validation, continuous effects and mixed initiative planning using PDDL. In *ICTAI*, pages 294–301. IEEE, 2004.

[Imai and Kishimoto, 2011] Tatsuya Imai and Akihiro Kishimoto. A novel technique for avoiding plateaus of greedy best-first search in satisficing planning. In Wolfram Burgard and Dan Roth, editors, *AAAI*, pages 985–991, 2011.

[Karpas and Domshlak, 2009] Erez Karpas and Carmel Domshlak. Cost-optimal planning with landmarks. In *IJCAI*, pages 1728–1733, 2009.

[Lelis *et al.*, 2013] Levi HS Lelis, Sandra Zilles, and Robert C Holte. Stratified tree search: a novel suboptimal heuristic search algorithm. In *AAMAS*, pages 555–562, 2013.

[Likhachev *et al.*, 2003] Maxim Likhachev, Geoffrey J Gordon, and Sebastian Thrun. ARA*: Anytime A* with provable bounds on sub-optimality. *NeurIPS*, 16:767–774, 2003.

[Narayanan *et al.*, 2015] Venkatraman Narayanan, Sandip Aine, and Maxim Likhachev. Improved multi-heuristic A* for searching with uncalibrated heuristics. In *SOCS*, pages 78–86, 2015.

[Pearl and Kim, 1982] Judea Pearl and Jin H Kim. Studies in semi-admissible heuristics. *TPAMI*, 4(4):392–399, 1982.

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

[Richter and Westphal, 2010] Silvia Richter and Matthias Westphal. The LAMA planner: Guiding cost-based anytime planning with landmarks. *JAIR*, 39:127–177, 2010.

[Richter *et al.*, 2010] Silvia Richter, Jordan Thayer, and Wheeler Ruml. The joy of forgetting: Faster anytime search via restarting. In *ICAPS*, pages 137–144, 2010.

[Seipp and Röger, 2018] Jendrik Seipp and Gabriele Röger. Fast downward stone soup 2018. *IPC2018–Classical Tracks*, pages 72–74, 2018.

[Thayer and Ruml, 2011] Jordan T Thayer and Wheeler Ruml. Bounded suboptimal search: A direct approach using inadmissible estimates. In *IJCAI*, pages 674–679, 2011.

[Valenzano *et al.*, 2013] Richard Valenzano, Shahab Jabbari Arfaee, Jordan Thayer, Roni Stern, and Nathan Sturtevant. Using alternative suboptimality bounds in heuristic search. In *ICAPS*, 2013.

[Valenzano *et al.*, 2014] Richard Anthony Valenzano, Nathan R Sturtevant, Jonathan Schaeffer, and Fan Xie. A comparison of knowledge-based GBFS enhancements and knowledge-free exploration. In *ICAPS*, pages 375–379, 2014.

[Vernaza *et al.*, 2009] Paul Vernaza, Maxim Likhachev, Subhrajit Bhattacharya, Sachin Chitta, Aleksandr Kushleyev, and Daniel D Lee. Search-based planning for a legged robot over rough terrain. In *ICRA*, pages 2380–2387, 2009.

[Wilt and Ruml, 2012] Christopher Makoto Wilt and Wheeler Ruml. When does weighted A* fail? In *SOCS*, pages 137–144, 2012.

[Xie *et al.*, 2014] Fan Xie, Martin Müller, Robert Holte, and Tatsuya Imai. Type-based exploration with multiple search queues for satisficing planning. In *AAAI*, pages 2395–2402, 2014.

[Zeng *et al.*, 2015] Yifeng Zeng, Xuefeng Chen, Xin Cao, Shengchao Qin, Marc Cavazza, and Yanping Xiang. Optimal route search with the coverage of users' preferences. In *IJCAI*, pages 2118–2124, 2015.