Investigating Exploration Techniques in Anytime Heuristic Search

Dawson Brown (500780579)
dawson.brown@ryerson.ca

Abstract

Knowledge-free, randomized exploration techniques in heuristic search algorithms have proved to be very effective. Noting that trusting too fully the evaluation function in suboptimal search can land the search into large plateaus or otherwise away from simpler paths to the goal state, the hope is that by devoting some energy to exploring the state space and not just marching forward via the evaluation function, a solution might be yielded faster.

Anytime Heuristic Search combines the benefits of fast suboptimal search with optimal search algorithms. It employs an inadmissible heuristic to arrive at solutions quickly, while the admissible heuristic is used to prune the open list and slowly converge on the optimal solution; in this way it gets the best of both worlds–the optimal solution in the end, but fast suboptimal solutions in the meantime.

In this study, I seek to examine the benefits of employing knowledge-free, randomized exploration in an anytime context to see if exploration can help anytime search algorithms converge on the optimal solution faster, use fewer resources, or both. For this, I will use Anytime Weight A* as the baseline and implement two exploration techniques giving two Exploration Anytime Algorithms: ϵ-AWA* and ϵB-AWA* . Several experiments using the sliding tile puzzle are used to compare the three algorithms. These experiments, while preliminary, convincingly demonstrate that using knowledge-free randomized exploration can be effective in anytime search algorithms and more investigation into the combination of randomized exploration and anytime search is needed.

Introduction

Algorithms like A* which are guaranteed to return the optimal solution to a search problem are often untenable in large problem spaces due to the time and resource requirements. To address this, bounded suboptimal search algorithms, like Weighted A* (WA*) and satisficing search algorithms like GBFS, can run faster with lower resource consumption in exchange for returning suboptimal solutions. WA* works by weighting an admissible heuristic to create an inadmissible one, Hansen and Zhou took advantage of this to create Anytime WA* (AWA*) in which a solution found with the weighted, inadmissible heuristic can be slowly refined overtime with the admissible heuristic giving a sequence of ever better solutions; in this way they achieve both desired outcomes–fast suboptimal solutions, and, in time, the optimal solution (Hansen and Zhou 2007).

Suboptimal Search algorithms have been shown to benefit a great deal from knowledge-free randomized exploration. GBFS has been shown to improve from both random exploration and from exploring according to a type system (Valenzano et al. 2014)(Xie et al. 2014). Type-based exploration has also been extended to work in WA* (Cohen, Valenzano, and McIlraith 2021). Suboptimal search algorithms that employ some form of randomized exploration are operating under the assumption that the heuristic and evaluation function used can be led astray in many domains falling into large local minima that are not a part of simpler solutions that the evaluation function can't see. Exploration can mitigate this by devoting time to exploring outside of the states that would normally be expanded.

In this project, I wanted to examine anytime search algorithms that make use of various exploration techniques over the course of their search for the optimal solution. To start, I paired AWA* with ϵ-greedy exploration, the simplest exploration technique. I then created my own, very simple, exploration technique termed ϵB-greedy in which nodes are sampled for expansion non-uniformly according to a Beta distribution B. I then also paired ϵB-greedy exploration with AWA* . Taking standard AWA* as a baseline, I evaluated the three algorithms to see if exploration could be beneficial in anytime heuristic search. The main contributions of this project are the following:

1. I combine ϵ-greedy exploration with AWA* to give the algorithm ϵ-AWA* .
2. I introduce ϵB-greedy exploration and combine it with AWA* to give ϵB-AWA* .
3. I do an in depth comparison of the three algorithms, AWA* , ϵ-AWA* , and ϵB-AWA* , on the unit cost sliding tile puzzle using the Manhattan Distance Heuristic; I briefly look at the inverse cost sliding tile puzzle; and lastly I touch on the Correct Tile Placement Heuristic to see if exploration can help when a very uninformed heuristic is used.

The rest of this paper has the following structure: first I go over the relevant background including AWA* and various exploration techniques from the literature; I then describe in

detail the two algorithms ϵ-AWA* and ϵB-AWA* ; this is followed by the evaluation of three algorithms; lastly there's a brief discussion and conclusion.

Background

For this study, I drew on AWA* as well as concepts in knowledge free exploration techniques.

AWA*

Hansen and Zhou introduced a method for converting Weighted A* into an Anytime algorithm giving AWA* (Hansen and Zhou 2007). To do so, they noted three things that are required:

1. An inadmissible evaluation function for selecting nodes for expansion. In the case of AWA* , that's a weighted, admissible heuristic (Hansen and Zhou 2007).
2. The search must continue after a (likely suboptimal) solution is found so that improved solutions can be found (Hansen and Zhou 2007).
3. An admissible evaluation function which serves as a lower bound on the optimal solution, along with the upper bound being the cost of the best solution found so for; these bounds can be used to prune the open list until it's empty at which point the last found solution must be optimal (Hansen and Zhou 2007).

Algorithm 1 gives the details of the 'anytime' part of AWA* –the steps omitted at line 14 are those that are part of normal WA*.

Algorithm 1: AWA*

Require: $f'(n) = g(n) + wh(n)$, $f(n) = g(n) + h(n)$
1: $g(init) = 0$
2: $OPEN \leftarrow \{init\}, CLOSED \leftarrow \emptyset$
3: $best \leftarrow$ None, $f(best) \leftarrow \infty$
4: **while** $OPEN \neq \emptyset$ **do**
5: $n \leftarrow \arg\min_{n' \in OPEN} f'(n')$
6: **if** $f(n) < f(best)$ **then**
7: $CLOSED \leftarrow CLOSED \cup \{n\}$
8: **for** $n_i \in Succ(n)$ **if** $f(n) < f(best)$ **do**
9: **if** n_i is goal **then**
10: $f(n_i) \leftarrow g(n_i) + c(n, n_i)$
11: $best \leftarrow n_i$
12: Continue
13: **end if**
14: ⋮
15: **end for**
16: **end if**
17: **end while**
18: **return** $best$

This approach is largely agnostic to the internals of the search being done so long as nodes are explored in some best first ordering that allows for open list pruning (Hansen and Zhou 2007). This makes it possible to include randomized exploration techniques in AWA* .

Randomized Exploration

Knowledge-free, randomized exploration has shown promise in satisficing search algorithms, in particular GBFS (Valenzano et al. 2014). Here, Valenzano *et al.* describe ϵ-GBFS in which, with probability $(1-\epsilon)$, the node with the smallest heuristic value is chosen for expansion, otherwise, a node is chosen uniformly at random from the open list (Valenzano et al. 2014). In their experimentation, adding ϵ-greedy node selection showed considerable improvement in several domains (Valenzano et al. 2014).

Despite showing improvement in some domains, and its relative simplicity, strictly random exploration (with probability ϵ) runs into problems when large local minima or plateaus are encountered. This is because the open list will come to be dominated by that plateau and so sampling randomly will likely expand a node on the plateau which is in part what exploration is trying to avoid (Xie et al. 2014)(Cohen, Valenzano, and McIlraith 2021). To address this, Xie *et al.* proposed an exploration technique based on a type system giving Type-based exploration in which nodes are bucketed by g-cost and their heuristic value. During exploration, a bucket is first sampled uniformly at random and then a node within that bucket is sampled (Xie et al. 2014). This approach achieves a much better spread over the state space than simple ϵ-greedy sampling (Xie et al. 2014).

Cohen *et al.* showed that WA* can also fall prey to these large plateaus. To alleviate this, they developed a modified Type-based exploration technique that maintains the bounded suboptimal guarantees of WA* giving Type-WA* (Cohen, Valenzano, and McIlraith 2021).

Exploration in AWA*

I implemented two exploration techniques within the framework of AWA* . The general idea is that for each node expansion, there is some probability ϵ that instead of expanding the best node according to the weighted evaluation function $f(n)$, some other node, in a knowledge-free way, is chosen from the open list for expansion. The two proposed techniques are ϵ-greedy and ϵB-greedy .

ϵ-AWA*

ϵ-AWA* employs the ϵ-greedy exploration and works much the same as ϵ-GBFS (in an anytime context). With this technique, with probability ϵ, a node is chosen uniformly at random from the open list.

Using ϵ-greedy in AWA* simply means potentially selecting an exploratory node during the node selection phase on Line 5 in Algorithm 1. Replacing Line 5 with the procedure outlined in Algorithm 2 gives ϵ-AWA* , where `randomSample` samples uniformly at random from the open list, and `randrange` samples uniformly at random from the given range.

ϵB-AWA*

ϵB-AWA* employs ϵB-greedy sampling, which is a bit more involved than ϵ-greedy and is meant to address some of the shortcomings of ϵ-greedy sampling. Xie's Type-Based sampling, along with its use in WA* by Cohen *et al.*, serves as

Algorithm 2: $\epsilon-$AWA* node selection

$y \leftarrow$ randrange(0,1)
if $y \leq \epsilon$ **then**
  return randomSample($OPEN$)
else
  return $\arg\min_{x \in OPEN} f'(x)$
end if

the inspiration for the ϵB-greedy sampling devised for ϵB-AWA* –where the aim is to achieve a better spread over the state space during sampling.

Simply put, ϵB-greedy samples from the open list non-uniformly. It is assumed that the open list is stored as a minimum heap; then the sampling is done in two steps: first sample a row from the heap non-uniformly, second sample a node uniformly from that row. The non-uniform sampling of a row is done according to a beta distribution, $B(\alpha, \beta)$, hence ϵB-greedy . Looking at Figure 1, sampling a row can be thought of as evenly spacing the rows between 0 and 1 along the x-axis and then sampling one according to the probability density above it. The intuition behind this is that, while heaps don't make strong guarantees about the order of nodes, they do guarantee that a parent is larger than both its children. This means that, in general, sampling from higher rows will yield smaller f-cost nodes than sampling from lower rows. And so, if you bias your row sampling to middling or lower rows you should expect to see a better spread across the state space. Furthermore, I think it's a fair assumption that those nodes with smaller f-costs are likely to be expanded sooner in the usual way (i.e. by the selection done on Line 5 in Algorithm 1) regardless of plateaus, and so exploration should focus on those nodes that aren't close to conventional expansion. This sampling technique also benefits from being quite simple to implement.

Algorithm 3: ϵB-AWA* node selection

Require: $\gamma \leftarrow beta(\alpha, \beta)$
$y \leftarrow$ randrange(0,1)
if $y \leq epsilon$ **then**
  $row \leftarrow$ sampleRow($OPEN, \gamma$)
  $start \leftarrow 2^{row} - 1$
  $end \leftarrow 2 \cdot start$
  return randomSample($OPEN[start : end]$)
else
  return $\arg\min_{x \in OPEN} f'(x)$
end if

Much like in ϵ-AWA* , the two-step sampling of ϵB-greedy is employed in ϵB-AWA* at each expansion with probability ϵ. The procedure outlined in Algorithm 3 can be invoked instead of Line 5 in Algorithm 1 to get ϵB-AWA* .

Evaluation

In order to evaluate the usefulness of exploration in AWA* , a number of experiments were conducted on the sliding tile puzzle. All experiments are run on an 8-core 1.70GHz machine with 16 gigabytes of memory. For all experiments, the two Exploration Anytime algorithms will be compared to a standard AWA* implementation which will serve as the benchmark.

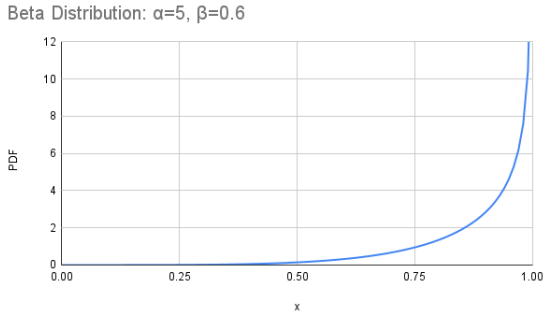

Figure 1: Beta Distribution used in ϵB-AWA*

Average Number of Incumbent Solutions					
Weight	AWA*	0.1-ϵ	0.3-ϵ	0.1-ϵB	0.3-ϵB
1.3	**2.24**	2.25	2.28	2.26	2.28
2	3.91	4.09	**3.75**	3.94	3.86
3	7.07	6.69	**6.07**	6.54	6.19
5	11.15	10.37	**9.28**	10.05	9.65
10	15.49	15.03	**13.27**	14.56	13.74

Table 1: Unit Cost: Incumbent Solutions

Experiment Setup and Overview

Experiments were conducted on one hundred instances of the unit cost 12-puzzle. The primary focus was on the Unit Cost Sliding Tile puzzle using the Manhattan Distance Heuristic. For this domain, 5 weights are considered for each algorithm, 1.3, 2, 3, 5, and 10. For ϵ-AWA* and ϵB-AWA* , two different values of ϵ are used, 0.1 and 0.3. Depending on the value of ϵ, ϵ-AWA* and ϵB-AWA* will be abbreviated to 0.1-ϵ, 0.3-ϵ, 0.1-ϵB, and 0.3-ϵB respectively. The beta distribution in all ϵB-AWA* instances is fixed at $\alpha = 5$ and $\beta = 0.6$.

There are also two small extensions to the unit cost experiments. In the first, the Inverse Sliding Tile Puzzle is briefly investigated using the Manhattan Distance Heuristic. In these experiments, the value of ϵ is fixed at 0.3, and only 3 weights are considered: 1.3, 5, and 10. The Inverse Sliding Tile Puzzle was chosen because it tends to be harder than the Unit Cost variant. The second extension involves degrading the heuristic and using the Correct Tile Placement Heuristic. This was done to investigate how exploration may help or hinder with a highly uninformed heuristic.

Unit Cost Tile Puzzle

In general, it was found that adding exploration to AWA* for the sliding tile puzzle proved to be very effective. Both ϵ-AWA* and ϵB-AWA* required fewer incumbent solutions before finding the optimal solution, as summarized in Table 1.

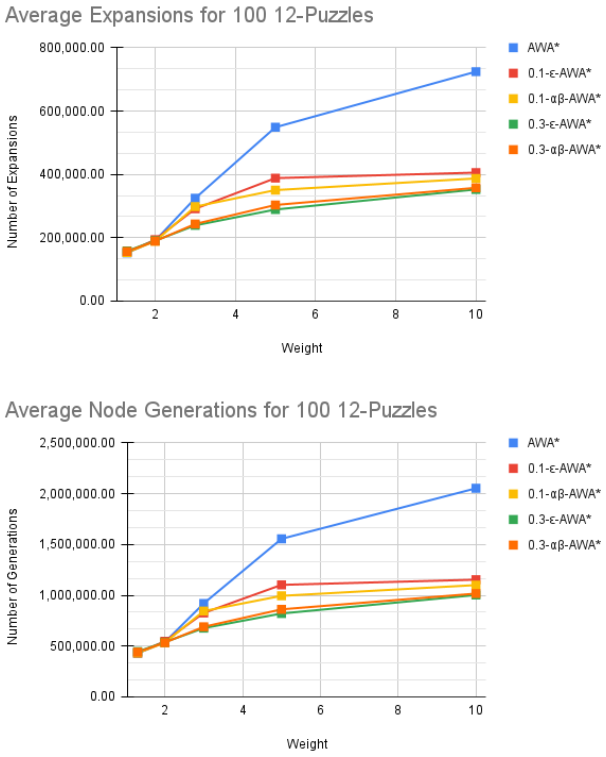

Figure 2: Unit Cost: Complexity

In all cases, except for weight=1.3, 0.1-ϵ-AWA* required the fewest incumbent solutions before finding the optimal. However, both exploration techniques were consistently better than standard AWA* by this metric.

To gauge the complexity of each algorithm, the number of expanded nodes and the total number of generated nodes are considered. By this metric exploration shows a huge improvement over standard AWA* , as shown in Figure 2.

Standard AWA* requires close to 800,000 expansions in the case of a weight of 10, while all four exploration algorithms require in the neighbourhood of 400,000. AWA* seems to be far more impacted by increasing the weight, and thereby degrading the accuracy of the evaluation function, than the ϵ and ϵB variants–which both require a similar number of expansions for a weight of 5 and a weight of 10. Perhaps most surprising is that setting ϵ to 0.3 gets the best results.

Adding to this, exploration in anytime sees an analogous improvement in its runtime when compared to standard AWA* , as seen in Figure 3. This is not guaranteed as it could be the case that the improved node selection procedure takes enough time so as to offset the time savings of expanding fewer nodes. In the cases of ϵ-AWA* and ϵB-AWA* , the node selection procedure is sufficiently simple that the expansion savings result in analogous time savings.

To summarize the above results, it's useful to look at how quickly each algorithm converges on the optimal solution when plotted against time and number of expansions. Given both the ϵ and ϵB variants required fewer incumbent solutions, fewer expansions to get those solutions, and took less

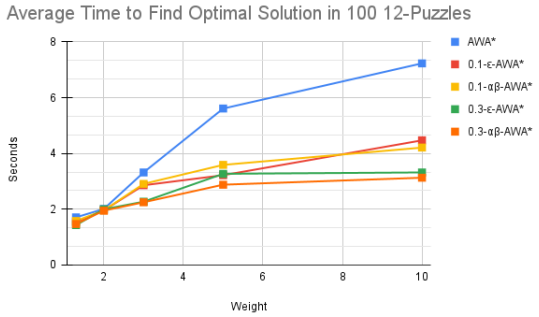

Figure 3: Unit Cost: Runtime

Average Number of Incumbent Solutions			
Weight	AWA*	0.3-ϵ-AWA*	0.3-ϵB-AWA*
1.3	3.14	3.13	**2.97**
5	21.91	15.66	**15.41**
10	30.8	**20.4**	21.26

Table 2: Inverse Cost: Incumbent Solutions

time, it should be the case that they approach the optimal solution much faster than AWA* , which is precisely what is seen in Figure 4.

Here, solution quality is $\frac{incumbent-cost}{optimal-cost}$ for a given problem and the solution qualities are averaged across all problem instances. For expansion convergence, the incumbent cost was polled every 50,000 expansions; for the time convergence the incumbent cost was polled every one second (notice the log x-axis in both charts). In this, only a weight of 10 and an epsilon value of 0.3 were considered.

Inverse Cost Tile Puzzle

Experimenting with the Inverse Cost Tile Puzzle involved the same set of 12-puzzle instances. For these experiments, only weights of 1.3, 5, and 10 were considered and ϵ was fixed at 0.3 because the $\epsilon = 0.3$ variants performed best in the unit cost problems.

Exploration also proved very beneficial in the Inverse Tile Puzzle. Table 2 shows the average number of incumbent solutions found before the optimal is found, and much like the unit cost case, ϵ-AWA* and ϵB-AWA* performed better in all cases and the discrepancy grew with the weight.

The complexity, again taken to be the number of expansions and number of generated nodes, painted a very similar picture to the unit cost case–with the caveat that each algorithm needed twice or more the number of expansions and generations that it did in the unit cost case. As shown in Table 3, as the weight increased, ϵ-AWA* and ϵB-AWA* required less than half the number of expansions than AWA* did. The difference is even more dramatic than the unit cost case, which was very surprising.

Similarly, for the number of generations, while the three algorithms were competitive with a low weight, as the weight increases, the performance of AWA* suffers far more dramatically than ϵ-AWA* and ϵB-AWA* .

The runtime of the algorithms, unsurprisingly since the

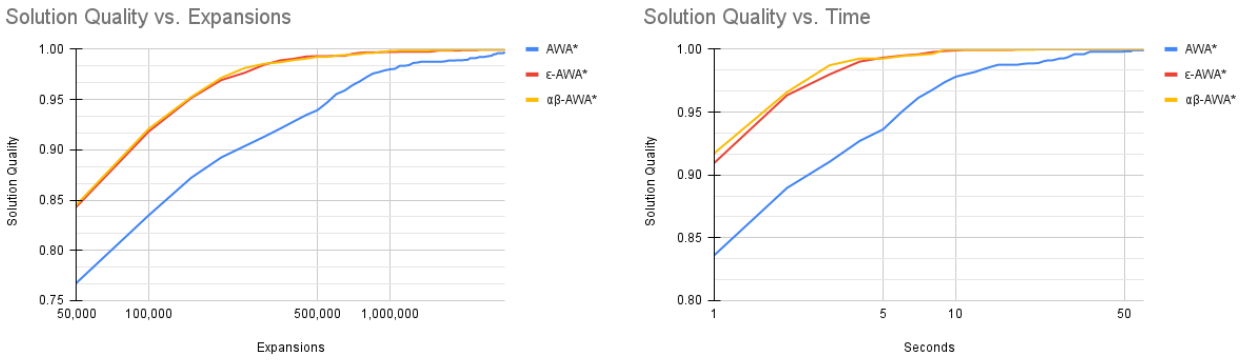

Figure 4: Unit Cost: Convergence

Average Number of Expansions			
Weight	AWA*	0.3-ϵ-AWA*	0.3-ϵB-AWA*
1.3	**186,038.15**	211,198.92	206,175.07
5	1,408,646.74	622,837.79	**587,692.64**
10	1,901,236.58	**709,049.28**	769,546.95

Table 3: Inverse Cost: Expansions

Average Number of Node Generations			
Weight	AWA*	0.3-ϵ-AWA*	0.3-ϵB-AWA*
1.3	**529,095.35**	603,706.60	589,437.60
5	4,003,219.50	1,780,788.61	**1,680,521.61**
10	5,401,559.63	**2,027,138.81**	2,200,575.94

Table 4: Inverse Costs: Generations

domain doesn't affect the time complexity of the sampling, also improved roughly proportionally to the number of required expansions. I believe this is for much the same reason as the unit cost case where since the exploration technique didn't result in much additional work, each saved expansion corresponds to an equivalent time saving.

All in all, I was very surprised and impressed with the efficacy of both exploration techniques in the Inverse Tile Problem. Because it is a harder problem with I believe fewer plateaus, I expected more modest results.

Degraded Heuristic in Unit Cost Sliding Tiles

Given the promising results above, it would be expected that when the heuristic gets degraded exploration should help even more, as degraded heuristics result in more plateaus and less confidence about the 'best' node in the open list. However, in looking at the summary evaluation of solution quality over time and expansions using the Correct Tile Placement Heuristic, this did not turn out to be case, as seen in Figure 5.

Average Number of Seconds to Find Optimal			
Weight	AWA*	0.3-ϵ-AWA*	0.3-ϵB-AWA*
1.3	2.25	**2.15**	**2.15**
5	16.38	6.65	**6.34**
10	21.57	**7.3**	8.09

Table 5: Inverse Costs: Runtime

Here, again, only a weight of 10 and an epsilon value of 0.3 were considered. For expansions, each algorithm was limited to 3,000,000, and for runtime each algorithm was limited to one minute.

While in the beginning AWA* actually acquires the best incumbent solutions, they all more or less approach the optimal solution at the same rate. This seems to simply demonstrate just how bad the Correct Placement Heuristic is. I think what is going on here is that the Correct Placement Heuristic is so bad it affectively adds and pulls nodes from the open list at random and so adding an exploration technique in no way helps since choosing the 'best' node is also, in effect, exploratory.

Discussion and Conclusion

While the results above are promising, the experimentation done was very limited in its scope and left a lot to be desired.

For starters, very few domains were investigated. Just investigating the tile puzzle makes it difficult to make any general statements about these results. In particular, domains that exhibit large uninformed heuristic zones and dead-ends would be very interesting to look at because that's where exploration helps the most.

The investigation into ϵB-AWA* was premature and incomplete. The ϵB-greedy sampling technique should be examined in standard suboptimal search before looking at it in anytime search. Discovering any merits or shortcomings of ϵB-greedy sampling would require a whole set of experiments examining different distributions across various domains using different suboptimal search algorithms. Only then should an in depth study into ϵB-greedy and anytime search be done. That said, based on these early results, I'd say ϵB-AWA* looks promising, but the fact it was not convincingly better than ϵ-greedy sampling means more experimentation needs to be done.

There are many other exploration techniques that could be evaluated in an anytime context. Type-Based exploration, which appears to perform very well in many domains would be an obvious starting point.

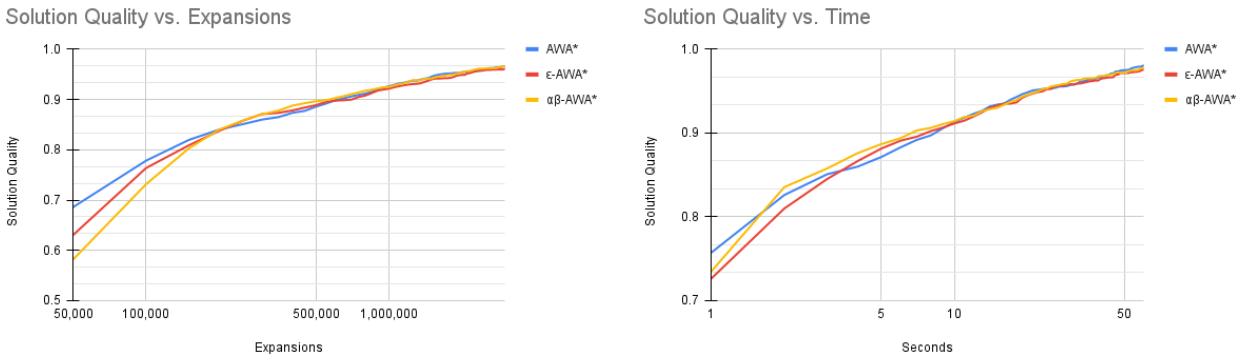

Figure 5: Unit Cost: Convergence with Degrade Heuristic

Lastly, the results of exploration in the Inverse Cost Tile Puzzle were very surprising and definitely worth a closer look.

Ultimately, I think this was a useful pilot study into using randomized exploration in an anytime search algorithm, but still more investigation is required before anything definitive is said.

References

Cohen, E.; Valenzano, R.; and McIlraith, S. A. 2021. Type-WA*: Using Exploration in Bounded Suboptimal Planning. In *IJCAI*.

Hansen, E. A.; and Zhou, R. 2007. Anytime heuristic search. *Journal of Artificial Intelligence Research*, 28: 267–297.

Valenzano, R. A.; Sturtevant, N. R.; Schaeffer, J.; and Xie, F. 2014. A comparison of knowledge-based GBFS enhancements and knowledge-free exploration. In *Twenty-Fourth International Conference on Automated Planning and Scheduling*.

Xie, F.; Müller, M.; Holte, R.; and Imai, T. 2014. Type-based exploration with multiple search queues for satisficing planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 28.