Dynamic Exploration in Type-GBFS

Adam Sorrent[*], **Geerthan Srikantharajah**[*],

Toronto Metropolitan University
{adam.sorrenti, gsrikantharajah}@torontomu.ca

Abstract

Several recent heuristic search algorithms have focused on methods which systematically disregard the heuristic function. While these popular search algorithms report increases in performance without a significant increase in computational cost, they all use rudimentary techniques to determine when to disregard heuristics. We create a framework on top of Type-GBFS in the Fast Downward Planning System, allowing us to dynamically update the probability of ignoring the heuristic. We test our proposed dynamic exploration control using cached values to evaluate the potential of our method. While we do not see significant benefits over traditional Type-GBFS within our experiments, our tests only focus on a limited scope. We note that dynamic exploration control has a high theoretical potential. We provide a framework for others to test with. See our Fast Downward Planning System fork on GitHub[1].

Introduction

The basis of efficient search algorithms involves prioritizing which nodes to explore using some form of distance metric. Greedy best-first-search (GBFS) (Doran and Michie 1966) is one of these algorithms. GBFS uses a search heuristic, which estimates the distance to the goal node from any other node. To be specific, GBFS is a heuristic search algorithm which chooses the next reachable node with the lowest heuristic value in all cases (a greedy approach). If the calculated heuristic values are perfect, the algorithm works extremely well. However, heuristic values are estimates of the actual distance to the goal node, in order to save on computational requirements. A common issue with GBFS is the issue of local minima. Sometimes, the heuristic can mislead the search algorithm by suggesting that a certain section of the graph is valuable to explore (if it has the lowest heuristic value), even if the path is inefficient or leads to a dead end. While GBFS is very good at approaching problems when equipped with a good heuristic function, there are certain exceptions to the rule. Some techniques have been proposed to mitigate this issue (Felner, Kraus, and Korf 2003; Sturtevant, Valenzano, and Schaeffer 2013; Xie et al. 2014). The basis of these algorithms involves deciding when to trust the heuristic values (using GBFS), and falling back to a different strategy when needed. Ignoring the heuristic-based search algorithm when stuck at a local minima is beneficial, but it is difficult to correctly identify when GBFS is stuck at a local minima. Our paper focuses on the decision between using GBFS and an alternate strategy. While Type-GBFS and ϵ-GBFS both spend a constant amount of time on random exploration, our paper aims to dynamically decide which search algorithm to prioritize.

In our investigation, we introduce a strategy that dynamically adjusts the balance between Greedy Best-First Search (GBFS) and Type-Based Random Exploration depending on real-time search performance. This method leverages performance metrics from previous search steps to adjust the probability of choosing between the exploration strategies, aiming to optimize the search process effectively. Our experimental results reveal that while our dynamic approach, labelled as W-Random, competes closely with standard GBFS implementations in terms of the number of problems solved, it does not consistently surpass the static exploration strategies in terms of overall efficiency and solution quality. Specifically, W-Random solved 369 out of 414 problems, marginally less than the Default strategy's 371, but with a higher computational cost. These findings suggest that while the adaptive strategy has potential, its execution needs refinement to enhance both the efficiency of node exploration and the quality of the resultant paths.

Related Work

The most basic version of greedy best-first search is a suboptimal search algorithm which prioritizes the heuristic function first and foremost. Within the one open list of nodes that GBFS holds, the node with the lowest heuristic value is expanded, with its children added to the open list. Solely using a heuristic function differs from the popular A* family of algorithms, which use a combination of distance travelled (g) and the heuristic value (h). Suboptimal search algorithms within the A* family, such as weighted A*, do still prioritize the heuristic value, but the distance travelled is not ignored. Increasing the focus on the heuristic value leads to potentially getting misled, as heuristic values are typically generated as imperfect and fast calculations. An example of a problematic graph/heuristic combination can be seen in Figure 1, where the topmost node is the start node and the bottom-rightmost node (with $h = 0$) is the goal node. While

[*]These authors contributed equally.

[1]https://github.com/mbrotos/downward-dynamic-explore

the goal node is very easily accessible if the rightmost nodes are expanded, GBFS will prioritize the rest of the graph, seeing the (misleading) better heuristic values. At a small scale like this, there is not a major time loss, but this example is scalable; there could be a million nodes with a heuristic cost of 9 (instead of only 3, as shown in the example), and GBFS would have to evaluate every single one before getting to the correct path.

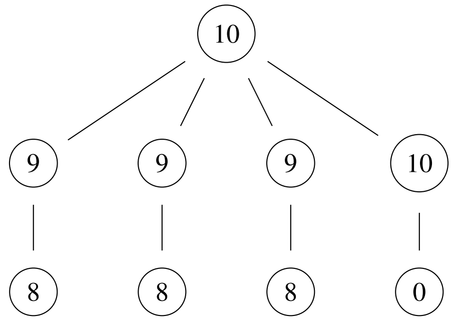

Figure 1: A tree showing the risk of a misleading heuristic. The value in each node represents its heuristic value. The uppermost node is the start node, and the node with $h = 0$ is the end node.

Several works have attempted to mitigate this issue. We know that GBFS can get stuck exploring less important local search spaces, exhibiting behaviour similar to depth-first search. GBFS will fully explore deep subtrees first, even if they are irrelevant, if the heuristic values deem the nodes to be relevant. K-BFS (Felner, Kraus, and Korf 2003) solves this by exploring the best k nodes simultaneously, prioritizing breadth instead of depth.

There are a few algorithms which fall back to random exploration periodically during the search process. The reasoning for this is shared across all of them: to prevent a misleading heuristic from trapping the search in a local minima. Random exploration simply means picking a random accessible node from the open list to expand. One very basic example is ϵ-greedy node selection (Sturtevant, Valenzano, and Schaeffer 2013). When using ϵ-greedy node selection, a constant ϵ is set before runtime where ϵ is between 0 and 1. The constant ϵ is used to decide when to explore with GBFS, and when to explore randomly. To be specific, the probability of expansion with GBFS is $(1 - \epsilon)$, and the probability of random expansion is ϵ. It is a very simple method both in terms of implementation and idea, yet in comparison to regular GBFS on the 2006, 2008, and 2011 International Planning Competition tasks, it solves 11% more problems.

Another example of a method which uses exploration is Type-GBFS (Xie et al. 2014). We know that GBFS gets stuck when in local minimas, clusters of similar nodes which mislead the heuristic. Type-GBFS introduces a type system to break up these clusters of similar nodes. Types are defined based on node characteristics - for example, any heuristic value or combination of heuristic values would be eligible. The types are only used to split up the ”random” exploration. Instead of randomly picking nodes from the entire open list, there is a separate bucketed list of nodes (bucketed based on their type). In each random exploration step, a random bucket is selected. Within the selected random bucket, a random node is selected and expanded. This type system replaces the standard random exploration seen with ϵ-greedy node selection, but the ϵ parameter still remains - the probability of GBFS being used is $(1 - \epsilon)$, and the probability of type-randomized exploration being used is ϵ. Type-GBFS is shown to solve more problems than GBFS on seven sets of International Planning Competition(Fox and Long 2011) tasks. Type-GBFS also solved more problems than ϵ-GBFS on IPC-2011 Nomystery. It is mentioned that Type-GBFS does not dominate regular GBFS, meaning that there are certain problems in which GBFS outperforms Type-GBFS. This is most likely due to random exploration being unnecessary for these problems, or within certain sections of the problems.

A different solution which performs well is Diverse BFS (Imai and Kishimoto 2011). DBFS operates a little differently compared to Type-GBFS and ϵ-GBFS. DBFS employs a global open list as well as a local open list. After selecting a node from the global open list to act as a start node, it performs regular GBFS using the local open list. Once regular GBFS is complete, the expanded nodes are added to the global open list, and the cycle continues. DBFS employs a parameter which allows it to explore closer to the global start node, instead of continually at the edges of the search space (like traditional GBFS). Another parameter is used to dictate how sub-optimal the selected node from the global open list should be, measured by its heuristic value. When both of these parameters are zero, regular GBFS is performed. DBFS shares similarities to Type-GBFS in that they both employ strategic random exploration.

Methods

Both ϵ-greedy node exploration and Type-GBFS rely on a random probability (ϵ) to determine when to perform random exploration, and when to perform regular GBFS. Xie *et al.* also note that there are cases in which regular GBFS outperforms Type-GBFS, potentially due to excessive random exploration. Instead of leaving the choice of random exploration as a constant probability, we create a framework which allows us to dynamically change the probability of performing random exploration.

Type-based random exploration, as seen in Type-GBFS, was shown to perform better in comparison to standard random exploration. Type-based random exploration is built to prevent getting stuck within clusters of misleading nodes, while standard random exploration is not. As a result, we use Type-GBFS as a baseline for our method instead of ϵ-GBFS.

We perform all of our testing on the Fast Downward Planning System (Helmert 2011). This is a heuristic search planning system built on C++ and Python 3. The Fast Downward Planning System includes many problem sets from the International Planning Competitions, and provides a framework to efficiently run many popular search algorithms while being able to make changes.

A basic version of our algorithm's implementation in the Fast Downward Planning System can be seen in Algorithm 1. This basic version uses the heuristic value in order to determine what algorithm to choose. In order to analyze the performance of random exploration (where type-random refers to type-based random exploration) and regular GBFS, we store a cache of past data for each search algorithm. We also set a constant learning rate before the start of the search. Upon the start of each search step, the LearnedRandom function is run. The heuristic value of the node is obtained, and stored in its relevant cache (either GBFS or type-random). The caches are size limited, so they only store recent information. After this step, we average the values stored in each cache, and identify which algorithm contains the lowest average heuristic value. This indicates that this algorithm is picking better nodes, with a lower distance to the goal. We increase the probability of this algorithm being picked by a factor of learning_rate. Over the course of the search, the method may decide to use GBFS more often, or type-random exploration more often.

Algorithm 1: GBFS and Type-Based Random Exploration Caching and Selection Algorithm

```
 1: Initialize GBFS cache with size $x$
 2: Initialize type-random cache with size $x$
 3: Set $learning\_rate$
 3: **function** LEARNEDRANDOM($stateID$)
 4: $node \leftarrow$ get_cached_estimate($stateID$)
 5: **if** $is\_dead\_end$(node) **then**
 6:     **continue**
 7: **end if**
 8: **if** node was selected using GBFS **then**
 9:     Add $g$ and $h$ values to GBFS cache
10: **else if** node was selected using type-random **then**
11:     Add $g$ and $h$ values to type-random cache
12: **end if**
13: $avg\_gbfs \leftarrow$ average of GBFS cache
14: $avg\_type\_rand \leftarrow$ average of type-random cache
15: Choose lower of $avg\_gbfs$ and $avg\_type\_rand$
16: Adjust selection probability by $learning\_rate$ {e.g.,
    50-50 becomes 50.01-49.99 with learning rate adjustment}
17: **return**
```

Results

This section presents the empirical results of our alternation strategy with relevant baselines. These experiments use a consistent set of planning problems from 10 unique domains shown in Table 1 with a total problem count of 414. All of these problems were derived from previous years International Planning Competitions (IPCs)(Fox and Long 2011).

The following presents four distinct search and exploration strategies:

- **Baseline**: Standard GBFS with any exploration.
- **Default**: Standard Type-GBFS with the default alternation list strategy.
- **Random**: Type-GBFS with a fully random alternation list strategy.
- **W-Random**: Type-GBFS with a dynamically updated weighted random alternation list strategy (as described in Algorithm 1).

Domain	Index	# of Problems
blocks	0	35
freecell	1	80
labyrinth-opt23-adl	2	20
logistics00	3	28
miconic-fulladl	4	151
nomystery-opt11-strips	5	20
parking-opt11-strips	6	20
parking-opt14-strips	7	20
recharging-robots-opt23-adl	8	20
visitall-opt14-strips	9	20
Total	–	414

Table 1: Mapping of domain names to an index for later reference and number of problems in each domain.

Experimental Setup

The experimental setup for our study is shown in Figure 2 and described as follows: Each problem was subjected to an overall time limit of five minutes, consistent across all experiments. The primary heuristic deployed was the FF heuristic, accompanied by the G heuristic tailored for the type system. Our search strategy involved a greedy best-first search, encapsulated within the "eager" main wrapper. This utilized an "alt" alternation open list, which allows selection from multiple open lists. Within this structure, two distinct strategies were included: a "single" implementing an open list for the FF heuristic and a "type_based" open list, which employs heuristic and g-cost binning. Note that the "type_based" open list is omitted in the Baseline experiment in order to replicate standard GBFS. A custom parameter, "decision," was implemented to switch between alternation strategies: '0' for the default strategy native to Fast Downward which evenly alternates, '1' for a completely random alternation, and '2' for a dynamically updated weighted random alternation list strategy. The "probs" parameter provided a list of floating point numbers representing a probability distribution across the open lists used in "alt."

The choice of domains for our experiments was influenced by previous work on Type-GBFS (Xie et al. 2014), maintaining a balance between the constraints of time and computational resources. The experiments were conducted on a machine equipped with a Ryzen 9 3900X CPU and 16GB of RAM, totalling approximately thirty minutes of runtime, facilitated by GNU parallel (Tange 2021).

Problem Solving Performance

Examining the problem solving performance from Table 2, our W-Random strategy shows robust effectiveness, solving 369 problems, slightly behind the optimal strategy in this benchmark with Default at 371. W-Random surpasses the

```
fast-downward.py
  --overall-time-limit 5m
  $DOMAIN_FILE $PROBLEM_FILE
  --evaluator "hff1=ff()"
  --evaluator "hg=g()"
  --search "eager(
      alt([
            single(hff1),,
            type_based([hff1,hg])
          ],
          decision=2,
          probs=[0.5, 0.5]),
      preferred=[])"
```

Figure 2: Fast-Downward planning system configuration

Random strategy, which solves 368 problems. Our baseline solves the least problems, with a total of 360. These results illustrate W-Random's capability to approach the Default's high performance. However, W-Random did not perform significantly better than the Random strategy in a number of total problems solved. This suggests that our approach to weighted exploration did not improve much over random exploration in terms of problems solved.

	Solved			
Domain	Baseline	Default	Random	W-Random
0	**35**	**35**	**35**	**35**
1	78	**79**	77	78
2	1	**2**	**2**	**2**
3	**28**	**28**	**28**	**28**
4	135	**138**	**138**	137
5	14	**19**	**19**	**19**
6	**20**	**20**	**20**	**20**
7	**20**	**20**	**20**	**20**
8	**10**	**10**	9	**10**
Total	360	**371**	368	369

Table 2: Number of problems solved by each strategy for each domain. The table lists the number of problems solved by each strategy for each domain and the total number of problems solved by each strategy. See Table 1 for the total number of problems.

Efficiency in Evaluations

In examining the alternation strategies based on the number of evaluations (Table 3), several patterns emerge. The Baseline strategy typically utilizes fewer evaluations across most domains, suggesting an efficient but potentially shallow search depth. While it is true that the Baseline strategy offers fewer evaluations in smaller search spaces, this is offset by inefficiencies in more complex planning domains, leading to an average of 100932.95 evaluations. Conversely, the Default strategy incurs a higher computational cost on all of the simpler problems, yet it averages 32247.65 evaluations, showing a much stronger performance on the rest of the domains.

The Random strategy shows the lowest average number of evaluations at 27606.38, reflecting its potential to quickly bypass complex sections of the search space, albeit without consistently improved problem-solving outcomes. Lastly, although designed to optimize its exploration process dynamically, the W-Random strategy results in a higher evaluation count compared to Default and Random (39729.22 on average), suggesting an over-exploration issue. Despite its complexity, this strategy does not proportionally increase the number of problems solved, indicating a need for refinement to enhance its efficiency.

Overall, these results suggest potential room for improvement in the number of evaluations performed by GBFS and Type-GBFS. The reduction in average evaluation for the Random strategy is promising for the efficacy of dynamic exploration, particularly in large search spaces. Furthermore, in specific domains such as #4 ('miconic-fulladl'), W-Random performs the least evaluation and may indicate efficacy for specific domains.

	Evaluations			
Domain	Baseline	Default	Random	W-Random
0	13920.80	**11069.63**	17900.86	19105.03
1	9046.35	11201.44	**6993.84**	14935.24
2	**207.00**	3184.00	729.50	1026.50
3	**870.54**	1574.64	1556.21	1525.04
4	14185.67	17100.22	15731.79	**11706.72**
5	883634.79	120003.00	**112224.37**	210321.58
6	**7250.50**	13426.25	11777.65	10217.05
7	**8799.70**	12163.70	11293.30	14581.20
8	**16646.80**	36673.00	42875.22	34485.00
9	**54767.37**	96080.65	54981.05	79388.80
Average	100932.95	32247.65	**27606.38**	39729.22

Table 3: Number of evaluations by each strategy for each domain. The table lists the average number of evaluations by each strategy for each domain and the total average number of evaluations by each strategy.

Plan Quality

When analyzing the quality of plans produced by each strategy (Tables 4 and 5), both the plan length and plan cost are key indicators. The Baseline strategy typically achieves shorter and less costly plans, suggesting that while it may explore more paths, it often directly identifies efficient solutions, as evidenced by the lowest average plan length of 79.75 and the lowest average plan cost of 79.09.

The Default strategy, while solving the most problems, tends to generate longer and more costly plans (averages of 90.26 for length and 88.95 for cost), indicative of its more exhaustive search and potential exploration of less direct paths to the solution. This could be attributed to its systematic yet broader exploration approach, which, while effective in problem-solving, may not always yield the most streamlined solutions.

The Random and W-Random strategies show similar patterns in plan quality, with slightly less optimal results compared to the Baseline but better than the Default strategy. This indicates a balanced approach in these strategies between thorough exploration and direct problem-solving, yet they do not consistently achieve the efficiency of the Baseline strategy. The W-Random strategy, specifically designed to dynamically adjust exploration efforts, does not significantly outperform the simpler Random strategy in terms of plan quality, suggesting that the dynamic adjustments may not always align with the most cost-effective or shortest paths to solutions.

Overall, the findings on plan quality bring to light the inherent trade-offs present in different search strategies. Strategies that are adept at solving more problems might generate less efficient plans, reflecting a strategic decision between breadth of problem-solving ability and depth of search efficiency.

	Plan Length			
Domain	Baseline	Default	Random	W-Random
0	75.54	**63.26**	67.60	67.77
1	**56.58**	60.00	57.19	58.40
2	**8.00**	17.50	13.50	13.50
3	**41.79**	41.89	42.18	41.82
4	**43.49**	45.01	44.67	44.46
5	**23.07**	26.21	26.26	26.21
6	43.75	43.75	42.80	**42.35**
7	42.15	**41.70**	43.05	43.80
8	15.30	15.70	**14.56**	15.70
9	**447.84**	547.55	510.70	532.20
Average	**79.75**	90.26	86.25	88.62

Table 4: Average plan length by each strategy for each domain. The table lists the average plan length by each strategy for each domain and the total average plan length by each strategy.

	Plan Cost			
Domain	Baseline	Default	Random	W-Random
0	75.54	**63.26**	67.60	67.77
1	**56.58**	60.00	57.19	58.40
2	**5.00**	8.00	7.00	7.00
3	**41.79**	41.89	42.18	41.82
4	**43.49**	45.01	44.67	44.46
5	**23.07**	26.21	26.26	26.21
6	43.75	43.75	42.80	**42.35**
7	42.15	**41.70**	43.05	43.80
8	11.70	12.10	**11.22**	11.80
9	**447.84**	547.55	510.70	532.20
Average	**79.09**	88.95	85.27	87.58

Table 5: Average plan cost by each strategy for each domain. The table lists the average plan cost by each strategy for each domain, as well as the total average plan cost by each strategy.

Summary

In summary, we evaluated various search strategies within the Greedy Best-First Search (GBFS) framework, focusing on their performance across a range of domains from the International Planning Competitions. Our newly introduced dynamic exploration algorithm, W-Random, aimed to adjust exploration strategies based on real-time performance metrics. While it performed competitively by solving 369 problems, slightly less than the Default strategy's 371, it did not significantly outperform other strategies in terms of efficiency or plan quality. The Baseline strategy, despite solving fewer problems, often produced more efficient plans, highlighting a trade-off between exploration depth and solution efficiency. The findings suggest that while dynamic adjustments in exploration strategies are promising, their implementation needs careful tuning to optimize both problem-solving capabilities and resource efficiency. Furthermore, our results highlight the complexities of balancing effective problem-solving with search efficiency in heuristic-driven search algorithms.

Future Work

Building upon the foundational work presented in this research, several avenues remain open for exploration to enhance the dynamic exploration control in heuristic search algorithms. The primary goal moving forward is to refine and expand the dynamic decision-making framework to better handle various search scenarios, particularly those involving complex and high-dimensional spaces.

Advanced metrics for decision-making are crucial; further research is needed to develop more sophisticated metrics that can dynamically gauge the effectiveness of exploration strategies. Metrics such as the distribution of heuristic values across the search frontier or the variability of path costs could provide deeper insights into the search process. Additionally, exploring the integration of machine learning models, such as regression models or neural networks, could embed past search metrics to predict the most effective exploration strategy at different points in the search process.

While this study focused on Greedy Best-First Search (GBFS) and its variants, applying the principles of dynamic exploration control to other search algorithms like A* could broaden the applicability of the research. Given the generalized architecture of our implementation, this work can serve as a framework for others to explore dynamically alternating open lists in many adjacent research problems. Moreover, conducting extensive empirical evaluations across a wider range of random seeds can help increase the statistical significance of our methods.

By pursuing these directions, we aim to refine the understanding of heuristic search exploration dynamics and enhance the utility and applicability of these methodologies. This approach is expected to lead to more robust, efficient, and adaptable heuristic search solutions, paving the way for their application in more complex and varied problem settings.

Acknowledgments

We would like to extend our gratitude to Professor Rick Valenzano at Toronto Metropolitan University, for his invaluable feedback on our research. We are thankful for his guidance and support throughout this project.

References

Doran, J. E.; and Michie, D. 1966. Experiments with the Graph Traverser Program. *Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences*, 294(1437): 235–259.

Felner, A.; Kraus, S.; and Korf, R. 2003. KBFS: K-best-first search. *Annals of Mathematics and Artificial Intelligence*, 39.

Fox, M.; and Long, D. 2011. The 3rd International Planning Competition: Results and Analysis. *CoRR*, abs/1106.5998.

Helmert, M. 2011. The Fast Downward Planning System. *CoRR*, abs/1109.6051.

Imai, T.; and Kishimoto, A. 2011. A Novel Technique for Avoiding Plateaus of Greedy Best-First Search in Satisficing Planning. *Proceedings of the International Symposium on Combinatorial Search*, 2(1): 197198.

Sturtevant, N. R.; Valenzano, R.; and Schaeffer, J. 2013. Adding Exploration to Greedy Best-First Search. *ERA*.

Tange, O. 2021. GNU Parallel 20210822 ('Kabul').

Xie, F.; Mller, M.; Holte, R.; and Imai, T. 2014. Type-Based Exploration with Multiple Search Queues for Satisficing Planning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 28(1).