Proceedings of the Twelfth International
Symposium on Combinatorial Search (SoCS 2019)

# A Case Study on the Importance of Low-Level Algorithmic Details in Domain-Independent Heuristics

**Ryo Kuroiwa, Alex Fukunaga**
Graduate School of Arts and Sciences
The University of Tokyo

In this paper, we show that low-level algorithmic details of domain-independent planning heuristics can have a surprisingly large impact on search performance. As a case study, we consider the well-known FF heuristic ($h_{ff}$) (Hoffmann and Nebel 2001).

A planning task $P = \langle V, s_0, s_*, A\rangle$, where $V$ is a set of variables, $s_0$ is the initial state, $s_*$ is the goal states, and $A$ is the set of actions. Each action $a$ has a cost $c(a)$, preconditions ($pre(a)$), and effects ($eff(a)$), and each effect $e$ has a effect condition ($cond(e)$) and a value assignment ($x \leftarrow v$). A plan for a planning task is an action sequence which makes $s_0$ transition to $s \in s_*$.

A well-known heuristic for satisficing planning is the FF heuristic (Hoffmann and Nebel 2001). $h_{ff}(s)$ is defined as the cost of a plan for a *relaxed planning task*, where multiple values can be assigned to a variable. As a side effect of computing $h_{ff}(s)$, a subset of applicable actions called "helpful actions" is obtained and exploited by search algorithms (Hoffmann and Nebel 2001; Helmert 2006; Nakhost and Müller 2013). Although the original $h_{ff}$ uses a planning graph to compute a relaxed plan (GRAPHPLAN) (Blum and Furst 1997; Hoffmann and Nebel 2001), Keyder and Geffner 2008 proposed a computation of $h_{ff}$ based on an additive heuristic $h_{add}$ (Bonet and Geffner 2001).

The widely used Fast Downward planning system (FD) (Helmert 2006; Richter and Westphal 2010) uses an $h_{add}$ based implementation. In FD (http://hg.fast-downward.org/), $h_{ff}$ is computed as follows: 1) decompose each action $a$ per each effect $e$ into unary actions $u$, such that $pre(u) = pre(a) \cup cond(e)$, effect $eff(u) = x \leftarrow v$, and cost $c(u) = c(a)$. 2) if an unary action $a$ is dominated by another action $b$, i.e. $eff(a) = eff(b) \wedge pre(b) \subseteq pre(a) \wedge c(b) \leq c(a)$, exclude $a$, 3) compute $h_{add}$ using Generalized Dijkstra algorithm (Liu, Koenig, and Furcy 2002) which maintains a priority queue, 4) and extract a relaxed plan and helpful actions. In step 3, FD uses an adaptive priority queue, which is a bucket based priority queue at first, but switches to the C++ Standard Library std::priority_queue when the priority of any entry exceeds 100. While tie-breaking of the bucket based priority queue is Last In First Out (LIFO), std::priority_queue is heap based and the ordering is not stable in GCC 5.4 (https://gcc.gnu.org/onlinedocs/gcc-5.4.0/libstdc++/manual/). Although the original $h_{ff}$ extracted helpful actions from the relaxed planning graph, step 4 restricts helpful actions to actions in the relaxed plan.

| | GBFS | | | +PO | | | MRW | | |
|---|---|---|---|---|---|---|---|---|---|
| | B | H | G | B | H | G | B | H | G |
| elevators | 16 | **17** | 15 | 17 | **19** | 18 | 20 | 20 | 20 |
| nomystery | 10 | 8 | **11** | 10 | 9 | **11** | 10 | 10 | **11** |
| parcprinter | **20** | 8 | **20** | **20** | 12 | **20** | 20 | 20 | 20 |
| pegsol | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 |
| scanalyzer | 17 | **18** | **18** | 18 | 18 | **20** | 17 | 17 | 17 |
| sokoban | **19** | 18 | **19** | 19 | 19 | 19 | 1 | 1 | 1 |
| tidybot | **16** | 14 | 13 | **15** | **15** | 13 | 17 | **18** | **18** |
| woodworking | **14** | 10 | **14** | **20** | 17 | **20** | 8 | 7 | **20** |
| barman | **3** | 0 | **3** | 9 | 3 | **17** | 19 | 19 | **20** |
| cavediving | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 |
| childsnack | 0 | 0 | 0 | **6** | **6** | 3 | 4 | **5** | 4 |
| citycar | 0 | 0 | 0 | **9** | 6 | 4 | 5 | 5 | **6** |
| floortile | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| ged | **20** | **20** | 19 | 20 | 20 | 20 | **20** | **20** | 15 |
| hiking | **19** | **19** | 11 | 19 | **20** | 12 | 18 | 18 | 18 |
| maintenance | 6 | 6 | **7** | 9 | **11** | 7 | **17** | **17** | 12 |
| openstacks | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 |
| parking | 4 | 7 | **8** | **13** | 10 | 5 | 6 | 0 | **20** |
| tetris | 10 | 14 | **16** | 18 | 17 | **19** | 8 | 8 | **17** |
| thoughtful | 8 | 11 | **12** | **14** | **14** | 12 | 20 | 20 | 20 |
| transport | 0 | 0 | 0 | **10** | 6 | 1 | 20 | 20 | 20 |
| visitall | 0 | **20** | 0 | 0 | **20** | 0 | 13 | 16 | **20** |
| agricola | 10 | 10 | 10 | **14** | 13 | 10 | 10 | 10 | 10 |
| caldera | 10 | 5 | **12** | 4 | 4 | **14** | 5 | 5 | **12** |
| caldera-split | 2 | **5** | 4 | 4 | **6** | 4 | 2 | 2 | 2 |
| data-network | 4 | **5** | **5** | 11 | **13** | 11 | **10** | **10** | 8 |
| flashfill | 8 | 8 | **10** | 13 | **14** | 10 | 9 | 9 | **12** |
| nurikabe | 10 | **13** | 9 | 9 | **10** | **10** | 14 | 14 | 14 |
| organic-synthesis | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| organic-synthesis-split | 8 | 9 | **11** | 8 | 9 | **11** | 4 | 4 | **5** |
| settlers | 3 | 3 | **4** | **14** | **14** | 11 | **20** | 17 | 17 |
| snake | 5 | 5 | 5 | 5 | 5 | **7** | 8 | **11** | 9 |
| spider | **11** | **11** | 10 | **14** | **14** | 13 | 11 | 12 | **14** |
| termes | **15** | 14 | 14 | **15** | 14 | 14 | **2** | **2** | 1 |
| total | 320 | 330 | **332** | 409 | **410** | 388 | 390 | 389 | **435** |
| diff (B,H) (H,G) (G,B) | 94 | 98 | 68 | 83 | 128 | 109 | 35 | 90 | 85 |

Table 1: Coverage comparation. diff(a,b) is the number of instances solved by either a or b and not solved by the other.

**Comparison of $h_{ff}$ Implementation Strategies** We evaluated a satisficing planner with 3 implementations of unit-cost $h_{ff}$: GRAPHPLAN (G), an $h_{add}$ based implementation with a bucket based priority queue (B), and an $h_{add}$ based implementation using a heap (H). We evaluated these 3 $h_{ff}$ implementations with 3 search strategies: Lazy Greedy Best First Search (GBFS), GBFS with helpful actions (+PO)

| | #min | | | #helpful | | |
|---|---|---|---|---|---|---|
| | B | H | G | B | H | G |
| woodworking | 1 | 0 | **17** | 1.140 | 1.214 | **19.933** |
| parking | 0 | 0 | **20** | - | - | - |
| tetris | 0 | 1 | **8** | 1.086 | 1.174 | **2.166** |
| visitall | 0 | 0 | 0 | **1.217** | 1.184 | 1.171 |
| caldera | 0 | 0 | **12** | 0.477 | 0.442 | **1.440** |
| flashfill | 2 | **4** | 0 | **0.720** | 0.605 | 0.432 |

Table 2: #min and #helpful per domains.

(Richter and Helmert 2009), and the Monte-Carlo Random Walk (MRW) algorithm used in Arvand-13 (Nakhost and Müller 2013). All of the algorithms were implemented in C++14 (GCC 5.4). Experiments were run on a machine with 16 cores (Xeon E5-2650 v2 2.60 GHz).For H, we used std::priority_queue, following FD. We use a wall-clock time limit of 30 min, a memory limit of 8 GB, and 34 domains from the satisficing track of IPC-11, IPC-14, and IPC-18 (20 instances/domain). For the domains with overlaps in IPC-11 and IPC-14, the IPC-14 version was used.

Table 1 shows the number of solved instances within resource limits (coverage). In GBFS and +PO, the coverage difference between B and H was >2 instances on parcprinter, woodworking, barman and visitall. In particular, B could not solve any visitall instances, while H solved all the instances. For MRW, large coverage differences between B and H were observed on parking, settlers, and snake.

For all search strategies, coverage using G differed by $\geq$3 from the coverage of $h_{add}$ based implementations in at least 5 domains. The total coverage of +PO with G performed worse than B and H. However, MRW with G solves more than 40 instances compared B and H.

In addition, Table 1 shows a diff(alg1,alg2) metric for each algorithm pair (B vs H, B vs G, H vs G, for GBFS, +PO, and MRW) which counts the number of instances solved by one algorithm but not the other. Although the total coverage of GBFS with B vs H differ only by 10, the diff(B,H) for GBFS is 94. Similarly, although total coverage of +PO with B vs H differ only by 1, diff(B,H) for +PO is 83. *Thus, the choice of priority queue tie-breaking policy causes significantly different search behavior, and the problems which are solved differ significantly depending on the policy.*

While diff(B,H) is <50%smaller than diff(H,G) and diff(B,G) for MRW, diff(B,H) is higher than diff(G,B) in GBFS – surprisingly the tie-breaking of the priority queue, a minor implementation detail of $h_{ff}$, sometimes has a larger impact than the method used to compute a relaxed plan.

Note that Arvand-13, the previous state-of-the-art MRW-based planner, is built on top of the FD codebase, and uses FD's $h_{add}$ based implementation of $h_{ff}$. MRW with B and H are competitive with the original Arvand-2013 (https://github.com/nhootan/Arvand2011) (coverage=360). *Using a planning-braph based $h_{ff}$ implementation resulted in a significant performance improvement compared to Arvand-13.*

One possible explanation for such a large performance gap is that different $h_{ff}$ variants compute diffrent $h$-values. Because $h$-values of $h_{ff}$ are the costs of relaxed plans, lower the $h$-value, the closer the relaxed plan is to the optimal relaxed plan. Table 2 shows the number of instances where the initial $h$-value of any $h_{ff}$ variant is strictly lower than all the others (#min) on 6 domains where MRW with G solved $\geq$3 instances more than B and H. While #min(G) > #min(B) and #min(H) on most of these domains, #min(G) was lower than #min(B) and #min(H) on visitall and flashfill.

Another possible cause of the large performance gap between G vs. B and H is the number of helpful actions. Action choices are biased by helpful actions in MRW, and helpful actions in $h_{add}$ based $h_{ff}$ are more restricted than in the original $h_{ff}$ because of step 2 and 4. Table 2 compares the mean # of helpful actions per state (#helpful) on instances solved by all 3 MRW variants. G generates more helpful actions than others on the domains except for visitall and flashfill.

It is possible that excluding actions in step 2 results in different relaxed plans and helpful actions, and both of these simultaneously affect search performance. We are currently continuing to investigate the cause of the performance gap.

**Conclusion** While the previous state-of-the-art MRW-based planner used the commonly used $h_{add}$ based $h_{ff}$ implementation (Nakhost and Müller 2013), we have shown that a planning-graph based implementation of $h_{ff}$ yields a surprisingly large performance improvement, resulting in an MRW satisficing planner which is significantly more competitive than MRW using an $h_{add}$ based $h_{ff}$ implementation. Furthermore, we showed that both the choice of relaxed plan computation method as well as the priority queue policy for $h_{add}$ significantly affect which specific instances are solved.

These results show that such seemingly minor "low-level algorithmic details" can have a significant impact on the performance of a heuristic. Although this paper evaluated $h_{ff}$, similar details may have a significant but overlooked impact in other heuristics (e.g., heuristics which embed or use $h_{ff}$).

## References


Blum, A. L., and Furst, M. L. 1997. Fast planning through planning graph analysis. *Artif. Intell.* 90(1):281 – 300.

Bonet, B., and Geffner, H. 2001. Planning as heuristic search. *Artif. Intell.* 129(1-2):5–33.

Helmert, M. 2006. The fast downward planning system. *JAIR* 26(1):191–246.

Hoffmann, J., and Nebel, B. 2001. The FF Planning System: Fast Plan Generation through Heuristic Search. *JAIR* 14:253–302.

Keyder, E., and Geffner, H. 2008. Heuristics for planning with action costs revisited. In *ECAI*, 588–592.

Liu, Y.; Koenig, S.; and Furcy, D. 2002. Speeding up the calculation of heuristics for heuristic search-based planning. In *AAAI*, 484–491.

Nakhost, H., and Müller, M. 2013. Towards a second generation random walk planner: An experimental exploration. In *IJCAI*, 2336–2342.

Richter, S., and Helmert, M. 2009. Preferred operators and deferred evaluation in satisficing planning. In *ICAPS*.

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