# Base Strategy Still Matters: Triangle Search in Domain-Independent Planning

**Jordan Thayer**[1], **Sofia Lemons**[2], **Jendrik Seipp**[1]

[1]Linköping University, Sweden
[2]Accenture, Indianapolis, IN, USA
jordan.thayer@liu.se, sofia@snlemons.com, jendrik.seipp@liu.se

**Abstract**

LAMA has long been a standard reference point for satisficing domain-independent planning, combining planning-specific enhancements such as landmarks, preferred operators, and multiple queues with search algorithms like restarting weighted A* and greedy best-first search. In this paper, we revisit triangle search as a base-level search strategy for planning and compare it directly against the strategies that commonly underlie modern planners. In controlled, head-to-head comparisons using the same heuristics with planning-specific enhancements disabled, triangle search consistently improves over greedy best-first search and rectangle search on the Autoscale satisficing benchmark suite. While LAMA performs best overall, our results suggest that search strategy alone has more room for improvement than is commonly assumed in planning. On its own, triangle search already reproduces some of the behavior of these enhancements, which motivates integrating them explicitly in future work.

## 1 Introduction

Heuristic search (Hart, Nilsson, and Raphael 1968) and domain-independent planning (Fikes and Nilsson 1971) remain central to AI because they let us solve hard problems with shared, reusable machinery. The modern planning ecosystem has made enormous progress on top of this foundation. However, in many high-performing planners the underlying search engine is one of a few classic choices, mainly especially greedy search and weighted-A* (Pohl 1970). Several variants on the latter are used, including Restarting Weighted A* (Richter, Thayer, and Ruml 2010), and enforced hill-climbing (Hoffmann and Nebel 2001),.

LAMA (Richter and Westphal 2010) is the best-known example: a strong baseline strategy combined with planning-specific enhancements such as landmarks (Hoffmann, Porteous, and Sebastia 2004), preferred operators (Richter and Helmert 2009), multiple queues (Röger and Helmert 2010), and boosting to serve from some queues preferentially (Richter and Westphal 2010). This design has remained competitive for more than fifteen years. At the same time, it makes it difficult to determine how much of the performance stems from the search strategy itself and how much from the planning-specific enhancements.

This paper focuses on that question. We revisit triangle search (Lemons et al. 2023), an anytime heuristic search algorithm introduced outside mainstream planning research, and ask how much can be gained by changing only the base search strategy while holding heuristic information fixed and disabling planning-specific search enhancements. We find that the choice of base strategy still matters considerably: on the Autoscale satisficing benchmarks (Torralba, Seipp, and Sievers 2021), triangle search consistently improves over greedy best-first search and rectangle search as base strategies. Beyond implementing triangle search in a modern planner and conducting a large-scale empirical evaluation, we also identify where the behavior of triangle search qualitatively resembles planning-specific enhancement mechanisms, and discuss how these enhancements could be integrated into triangle search explicitly.

## 2 Related Work

Heuristic search algorithms drive the vast majority of competitive planning systems, as demonstrated repeatedly at the International Planning Competition (IPC) (Hoffmann and Edelkamp 2005; Linares López, Celorrio, and Olaya 2015; Vallati et al. 2015; Taitler et al. 2024). Across these algorithms, we typically assume an initial state, a goal test, successor generation via applicable actions, and a heuristic guidance signal. The practical differences come from how candidate states are ordered, filtered, and expanded under that shared setup and can be grouped along several axes:

**Optimal & Satisficing** At one extreme, optimal search algorithms such as A* (Hart, Nilsson, and Raphael 1968) seek minimum-cost solutions; satisficing algorithms such as greedy best-first search (GBFS) (Doran and Michie 1966) prioritize finding any solution quickly.

**Complete & Incomplete** Complete methods can find a solution if one exists (or prove none exists in finite spaces); incomplete methods such as beam search (Bisiani 1987) and enforced hill climbing (Hoffmann and Nebel 2001) trade those guarantees for speed or focus.

**Single Solution & Anytime** Single-solution methods terminate after the first plan; anytime variants continue improving incumbent quality over time (Zilberstein 1996; Hansen and Zhou 2007).

**Tree Search & Graph Search** Tree-oriented methods reason over paths with limited duplicate handling, whereas graph-oriented methods explicitly manage repeated

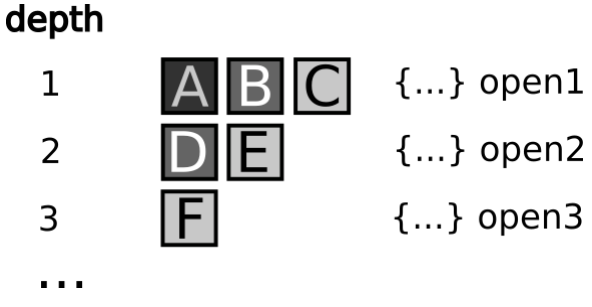


Figure 1: The exploration of triangle search with slope 1. Reproduced from Lemons et al. (2023).

states. In tightly connected state spaces, this distinction strongly affects practical behavior and resource usage.

Because planning problems are hard, empirical evaluations usually focus on coverage (the number of solved tasks) rather than completeness. Empirically, the complete algorithms we evaluate outperform the incomplete baseline. Anytime behavior also matters, but finding a first solution is already hard enough that first-solution performance gives a reasonable comparison of search strategies under resource limits.

## 2.1 Triangle Search and its Influences

Triangle search is a recently introduced anytime heuristic search algorithm (Lemons et al. 2023). Although designed as an anytime algorithm, it is straightforward to use as a satisficing method: if we stop at the first incumbent, triangle search is a first-solution search strategy with a different expansion order than standard global best-first baselines.

Triangle search, a relative of rectangle search (Lemons et al. 2024), can be seen as an iteratively widening and deepening beam search, conceptually resulting in a triangular shape. At each iteration, it is allowed to expand one additional node from each previously used depth level, and one node from a new depth level. Figure 1 demonstrates three iterations of triangle search. In the first iteration, node A is selected from $open1$ (the open list for nodes at depth 1) because it has the lowest $h(n)$ (or wins tie-breaking). It is expanded, and its children are inserted into $open2$. In the second iteration, another node (B) is selected and expanded from $open1$, its children are inserted into $open2$, and then a node (D) is selected from $open2$, with its children being inserted into $open3$. In the third iteration, node C is expanded from $open1$, node E is expanded from $open2$ and node F is expanded from $open3$, with its children being inserted into a new queue, $open4$ (not shown) for possible future iterations.

Lemons et al. (2023) study triangle search on classic search benchmarks and provide full pseudocode. Their evaluation left open the question of whether or not the strong performance of triangle search transfers to domain-independent planning; it does, as we see in our evaluation. This performance is driven by the design of triangle search, which sits on three separate ideas: search-order policy, frontier windowing/subsetting, and diversification pressure.

**Search-Order** Search order is an obvious and historically important design axis in heuristic search: changing the key function from $h(n)$ as in greedy best-first search to $f(n) = g(n) + h(n)$ as in A* or to $f'(n) = g(n) + w \cdot h(n)$ as in weighted A* can substantially change both first-solution speed and solution quality. We hold search-order fixed in our evaluation (minimum $h$ value first, breaking ties in favor of lower $g$).

Triangle search ranks nodes in the same way as GBFS, but still expands them in a different order, because at each decision point only nodes at the same depth compete. It changes which nodes compete and when, rather than how they are ranked.

**Windowing & Subsetting** A second design axis is to introduce structure into the search frontier so that only part of the candidate set competes at a given point. In planning systems, this is often expressed as multiple ordered queues with scheduling policies (Röger and Helmert 2010); in the broader heuristic literature, a restricted subset competes under a chosen ordering and is shifted or refreshed over time, as in K-best first search (Felner, Kraus, and Korf 2003), $A^*_\epsilon$-style focal selection (Pearl and Kim 1982), beam-style width restriction (Bisiani 1987), and Window A* and related windowed variants (Aine, Chakrabarti, and Kumar 2007).

The motivating intuition is either that subsetting can be fairer when heuristic values are noisy or weakly discriminative or that subsetting can force progress by constraining how long the search can remain in one region of the frontier. Triangle search fits this tradition: it leaves the scoring formula untouched and instead partitions the frontier by depth, so that each depth level competes separately.

**Diversification and Novelty** Subsetting comes with the same risks as over-reliance on a heuristic evaluation function. If we are not careful how we focus search on an area of the space we believe to be good, we can sacrifice completeness and performance. We can hedge against this by choosing a larger subset, incorporating multiple heuristics, returning to previous search regions routinely (as triangle search does), or by introducing explicit diversification pressure as novelty-based methods such as Iterative Width do (Lipovetzky and Geffner 2012). Triangle search introduces this diversification implicitly: because it cycles across depth levels, its frontier is more diverse than that of standard global best-first search.

## 2.2 Domain-Independent Planning Systems

Just as heuristic search algorithms are generic strategies that can be applied to a range of problems, domain-independent planning systems are general-purpose tools for representing and solving planning problems. Fast Downward (Helmert 2006) is one such planning system, which provides a common framework for heuristic computation, state representation, and search. This shared infrastructure lets researchers build on each other's work. However, years of sustained development also mean that the search-algorithm layer is often just one component in a larger system, and attribution of performance to any single component can be difficult.

LAMA (Richter and Westphal 2010) is the canonical example of what this system can achieve. It combines several enhancements: a landmark-count heuristic (Richter, Helmert, and Westphal 2008) paired with the FF heuristic (Hoffmann and Nebel 2001), which together exploit problem structure to generate complementary guidance signals; preferred operators (Richter and Helmert 2009), a generalization of helpful actions (Hoffmann and Nebel 2001), that bias expansion toward actions identified as locally promising; multiple heuristic queues with round-robin or boosting scheduling that let competing guidance signals each receive expansion budget; and a restarting weighted-A* schedule control (Richter, Thayer, and Ruml 2010) that modulates the cost-versus-speed tradeoff over time. Each enhancement addresses a known failure mode, and together they have kept LAMA competitive across many benchmark sets.

### 2.3 Heuristic Search as Planning Core

LAMA has heuristic search at its core. It first runs a greedy best-first search to find an initial solution, then improves that solution with a restarting weighted A* (RWA*) schedule: an anytime sequence of weighted A* searches whose weight $w$ is decreased toward 1. Because we compare first-solution performance, we are effectively comparing triangle search to LAMA's initial greedy best-first search, which breaks ties in favor of lower $g$ values.

LAMA, like most performant planners, relies on planning-specific mechanisms to improve performance. Two are of particular interest here, as they interact heavily with base-search behavior. Preferred operators identify a subset of applicable actions as locally promising, typically based on delete relaxation, and bias the search toward expanding states reached by those actions. Queue prioritization and scheduling determine how expansion budget (which queue is served at each step) is allocated across multiple ordered queues, each of which may incorporate different heuristic signals or different subsets of the frontier. Both mechanisms effectively reshape which states compete for expansion at any given step—the same structural question that distinguishes triangle search from GBFS.

## 3 Evaluation

Our evaluation addresses five questions: (Q1) whether triangle search improves coverage over unenhanced GBFS (the base search underlying LAMA's first iteration) when both use the same heuristic, (Q2) whether these gains are general or concentrated in a few favorable cases, (Q3) which runtime, expansion, and memory tradeoffs accompany them, (Q4) how large the remaining gap to fully enabled LAMA is, and (Q5) which planning-specific enhancements account for that gap when added back in targeted comparisons. We use hand-crafted illustrative examples to explain the behaviors we observe in aggregate. Coverage and the resources needed to find a first solution are the primary targets throughout.

### 3.1 Methodology

We implemented triangle search in the Fast Downward planning system (Helmert 2006) and used the Lab toolkit (Seipp et al. 2017) to run all experiments on the Tetralith compute cluster, where standard nodes have two Intel Xeon Gold 6130 CPUs (32 cores per node) and 96 GiB of RAM. All runs are single-core planner runs with per-instance limits of 15 minutes of CPU time and 8 GB of memory. The benchmark suite is the Autoscale satisficing set (Torralba, Seipp, and Sievers 2021).

For Q1 through Q3, we compare triangle search against GBFS and rectangle search. We do not run RWA* as a standalone configuration: it is the anytime backbone of LAMA, but LAMA finds its first solution with a greedy best-first search, so outperforming GBFS already lower-bounds what a full RWA* comparison would show. For Q4 and Q5, we turn on specific planning enhancements until we re-arrive at LAMA's first iteration. Across all runs, we hold fixed the non-search factors that would otherwise confound attribution: the heuristic given to each configuration, the per-instance limits, and the benchmark instances.

We label each configuration by its base algorithm, heuristic, and enhancement setting. The base algorithm is GBFS, triangle search (TRI), or rectangle search (RECT), and the prefix LAZY- marks lazy (deferred) heuristic evaluation. The heuristic is either $h^{\text{FF}}$ (FF) or $h^{\text{LM-count}}$ (LM). A PO suffix enables preferred operators with boosting; without it, the planning-specific enhancements are disabled. For example, LAZY-GBFS-FF-PO is GBFS with lazy evaluation, $h^{\text{FF}}$, and preferred operators, whereas TRI-LM is triangle search with $h^{\text{LM-count}}$ and no enhancements. LAMA denotes LAMA's first iteration, which combines all of these enhancements. In Tables 1 and 2 we drop the heuristic from the label, since each table fixes one heuristic.

The primary metric is coverage: instances solved within the per-instance limits. Secondary metrics are geometric-mean runtime, node expansions and generations, and peak memory, supporting Q3 by characterizing the cost of any coverage gains. We report results as overall aggregates and as per-domain deltas relative to LAMA.

### 3.2 Findings

When planning enhancements are disabled, triangle search consistently outperforms GBFS and rectangle search under both heuristic settings (Tables 1 and 2): the base search strategy alone makes a substantial difference in first-solution performance. The effect of the enhancements depends strongly on the heuristic: preferred operators and lazy evaluation yield large gains for GBFS with $h^{\text{FF}}$, but provide no benefit with $h^{\text{LM-count}}$, where the advantage of triangle search holds regardless of enhancement configuration. A substantial gap to LAMA remains, traced below to specific enhancements.

**Base Search Strategy Matters** Triangle search outperforms GBFS on coverage under both heuristic settings (Tables 1 and 2): with $h^{\text{FF}}$ and enhancements disabled the margin is 118 instances; with $h^{\text{LM-count}}$ it is 83. Rectangle search falls between the two, solving 12 more instances than GBFS, but 106 fewer than triangle search. The coverage improvement of triangle search comes at a cost. It expands fewer nodes per second than other algorithms. The cost is not per-heap maintenance: partitioning the frontier into one open

| Algorithm | Cov. | Exp. (100k) | Gen. (100k) | Sec. |
|---|---|---|---|---|
| LAMA | 878 | 0.42 | 4.24 | 0.06 |
| TRI | 576 | 0.85 | 14.15 | 0.04 |
| RECT | 470 | 5.08 | 40.10 | 1.64 |
| GBFS | 458 | 2.49 | 36.71 | 0.05 |
| LAZY-GBFS | 464 | 2.63 | 44.36 | 0.05 |
| GBFS-PO | 622 | 0.65 | 5.60 | 0.05 |
| LAZY-GBFS-PO | 682 | 0.56 | 4.67 | 0.04 |

Table 1: Summary statistics for search algorithms using $h^{\text{FF}}$, with LAMA as a point of comparison. Cov. is coverage (the number of solved instances); Exp. and Gen. are the mean numbers of expanded and generated nodes over solved instances, in units of 100k nodes; and Sec. is the geometric-mean search time in seconds. Where algorithms fail to solve the instance, they are billed for the full runtime cutoff.

| Algorithm | Cov. | Exp. (100k) | Gen. (100k) | Sec. |
|---|---|---|---|---|
| LAMA | 878 | 0.42 | 4.24 | 0.06 |
| TRI | 688 | 2.15 | 31.28 | 0.04 |
| RECT | 486 | 4.03 | 23.76 | 4.58 |
| GBFS | 605 | 4.09 | 26.36 | 0.05 |
| GBFS-PO | 593 | 3.18 | 19.04 | 0.07 |
| LAZY-GBFS | 597 | 4.67 | 66.07 | 0.06 |
| LAZY-GBFS-PO | 603 | 4.46 | 44.03 | 0.06 |

Table 2: Summary statistics for search algorithms using $h^{\text{LM-count}}$, with LAMA as a point of comparison. Columns are as in Table 1.

list per depth level makes each heap smaller, so individual heap operations are, if anything, cheaper than on GBFS's single global open list. The overhead is in managing the queues themselves. Each pass extends *openlists* with *slope* new lists and trims those left empty. A direct implementation of the pseudo code repeatedly constructs and destroys a large number of heaps. This queue churn, rather than the cost of any individual heap operation, accounts for the lower node throughput. The coverage improvement shows the tradeoff is worthwhile.

**Planning-Specific Enhancements Matter Too** The effect of planning enhancements on GBFS depends sharply on the heuristic. With $h^{\text{FF}}$, preferred operators and lazy evaluation deliver a large boost: the best enhanced configuration (LAZY-GBFS-FF-PO, 682) surpasses unenhanced triangle search (TRI-FF, 576) by over 100 instances.

With $h^{\text{LM-count}}$, the enhancement picture is strikingly different. Preferred operators, lazy evaluation and their combination all fail to improve over the unenhanced baseline (Table 2). Triangle search (688) leads the best enhancement-enabled configuration by 85 instances. With this heuristic, base search strategy is the dominant factor. Tables 3 and 4 show that configurations based on triangle search outperform enhancement-enabled GBFS variants in both breadth (more domains with positive deltas) and depth (larger per-instance gains).

| Algorithm | $\Delta$ Cov. | Improving Dom. | Improving Inst. |
|---|---|---|---|
| TRI-LM | -190 | 10 | 96 |
| TRI-FF | -302 | 7 | 40 |
| GBFS-FF | -420 | 1 | 4 |
| LAZY-GBFS-FF | -414 | 1 | 2 |
| GBFS-FF-PO | -256 | 4 | 6 |
| LAZY-GBFS-FF-PO | -196 | 5 | 8 |
| GBFS-LM-PO | -285 | 9 | 31 |
| LAZY-GBFS-LM-PO | -275 | 9 | 33 |

Table 3: $\Delta$ from LAMA is the coverage difference between the configuration and LAMA. Improving domains is the number of domains where the configuration has higher coverage than LAMA. Improving instances is the total number of instances where the configuration solves an instance that LAMA does not solve. A negative $\Delta$ paired with positive improving-domain and -instance counts marks a configuration that trails LAMA in total coverage yet solves instances LAMA cannot.

| Algorithm | Snake | NoMystery | Tidybot | Thoughtful | Parking |
|---|---|---|---|---|---|
| TRI-FF | 21 | 18 | 13 | 10 | 9 |
| RECT-FF | 0 | -11 | -2 | -6 | 12 |
| GBFS-FF | 0 | 2 | 2 | 11 | 12 |
| LAZY-GBFS-FF | 0 | 7 | 4 | 8 | 13 |
| GBFS-FF-PO | 0 | 0 | 2 | 3 | 12 |
| LAZY-GBFS-FF-PO | 0 | 7 | 8 | 4 | 13 |
| TRI-LM | 6 | 14 | 3 | 10 | -6 |
| RECT-LM | -19 | -11 | -13 | -6 | -4 |
| LAZY-GBFS-LM-PO | 2 | 8 | 3 | 5 | 7 |

Table 4: Per-domain coverage gain relative to LAMA for the five domains where triangle search shows the largest improvements, with selected enhancement-enabled GBFS configurations included for comparison. Values are instances solved by the configuration minus instances solved by LAMA in that domain; positive values indicate the configuration outperforms LAMA locally. These five domains drive most of the complementarity summarized in Table 3.

**Per-Domain Impact** While LAMA solves the most instances overall, aggregate coverage hides where other configurations are complementary to it.

The largest domain-level gains for triangle search are in Snake (+21), NoMystery (+18), Tidybot (+13), and Thoughtful (+10). More than half the total improvements concentrate in five domains, with Snake and NoMystery accounting for most of that mass. This concentration suggests that triangle search exploits particular kinds of search spaces rather than being uniformly better across all domains.

At the instance level, the asymmetry is also substantial: the strongest triangle search configuration solves 96 instances that LAMA does not, compared to 33 for the strongest GBFS baseline in this comparison. The sets are not fully subsumed, however: enhancement-enabled GBFS still solves two instances not covered by triangle search (one in TPP and one in Tetris). Accordingly, a portfolio that includes triangle search could improve coverage over LAMA by roughly 8%, and this margin may grow as planning-specific enhancements are integrated into triangle search.

**Enhancement Comparison** Table 4 also supports a targeted comparison on the domains where the gains of triangle search are most pronounced. Because Snake appears specific to triangle search, we set it aside and focus on NoMystery, Tidybot, and Thoughtful, where the comparison against enhancement-enabled GBFS is more informative.

In NoMystery, preferred-operator guidance has consistently high impact. This raises the question of how much of that benefit the base control policy of triangle search already captures implicitly, and how much explicit preferred-operator integration would add.

Tidybot shows a different pattern: $h^{\text{LM-count}}$ alone already improves over LAMA, yet triangle search still gains more than planning-enhanced GBFS and remains ahead of base GBFS. Triangle search is therefore more than an indirect proxy for known planning enhancements; it also helps where those enhancements are weak or counterproductive.

### 3.3 Uninformed Heuristic Region Behavior

An uninformed heuristic region is an area of the search space where $h$ does not decrease. GBFS is most vulnerable: without improving estimates, all frontier nodes are equally ordered and expansion degenerates toward breadth-first over the uninformed heuristic region. EHC is even more brittle: it advances only when a local breadth-first search finds a strictly better state, so a region with no improvement forces it to search exhaustively or fail. High-weight variants of weighted A* are also susceptible when $g$ grows slowly within an uninformed heuristic region, reducing the effective contribution of the cost term.

Several approaches address uninformed heuristic regions through alternative orderings. The FF planner (Hoffmann and Nebel 2001) uses a bounded breadth-first search to escape plateaus, and the Arvand planner (Nakhost and Müller 2009) makes Monte-Carlo random walks its primary exploration mechanism. Within the GBFS family, local exploration triggers a local search or random walk when the heuristic stops improving (Xie, Müller, and Holte 2014), and type-based exploration interleaves a queue that diversifies over node types (Xie et al. 2014). Iterative Width methods (Lipovetzky and Geffner 2012; Fickert 2020) abandon heuristic ordering in favor of novelty-based ordering, forcing structural diversity. Domain reformulation identifies and prunes swamp regions directly (Pochter et al. 2010).

Triangle search requires no explicit detection mechanism for uninformed heuristic regions. Its depth striation means that when frontier nodes share the same heuristic value, deeper nodes still receive expansion budget: each depth maintains its own queue, and every forward pass expands the head of every active queue, so a node competes for position only against same-depth peers. The slope parameter in TS allows it to continue to follow a heuristic-biased walk from the edge of its current frontier. It's not a random walk as some algorithms use, and it is not required to improve the heuristic value as in enforced hill climbing. Still, the forced progression toward deeper nodes provides systematic pressure toward the region's boundary that flat global queues do not. The per-domain results in nomystery and tidybot, where Triangle leads both unenhanced and enhancement-enabled GBFS, are consistent with this escape mechanism being active in those domains.

The algorithms under consideration differ in how they escape uniformed heuristic regions. With low-$g$ tie-breaking, GBFS and $wA^*$ flood outward from the region's entrance in near-breadth-first order, shallower states before any deep node at the edge of a region is reached; the exit is guaranteed, but only after the entire shallow frontier is processed. EHC behaves the same way: its breadth-first search for an improving state also floods the frontier. A random walk can reach the exit without flooding, but has no guarantee and may drift away on this unbounded region. Triangle search expands one node from every occupied depth level per pass, sampling states in concentric rings outward from the region's entrance. Because triangle search opens a new deepest level on every pass and services every active level, it cannot stay confined to the shallow layers: it keeps generating deeper states until the exit's parent is expanded3

### 3.4 Multiple Queues and Frontier Partitioning

Multiple-queue systems prevent any single global ordering from monopolizing the expansion budget by partitioning the frontier across queues with different filtration criteria. LAMA uses two such criteria: preferred operators, which filter nodes reached via heuristic-relaxation-identified helpful actions into a separate queue; and multiple heuristic orderings, which assign nodes to queues ranked by different evaluation functions such as $h^{\text{FF}}$ and $h^{\text{LM-count}}$. The scheduler interleaves service across these queues so that each perspective receives expansion budget, with boosting adjusting allocation based on recent progress.

The level structure of triangle search imposes a structurally similar partition through a different criterion: distance from the search root. The partition is not based on what is known about a node, but on where it sits in the search tree. Problem-based filtration requires that the relaxed solution reliably identifies useful nodes. When those signals are weak or counterproductive, the queues filter on noise. Separating nodes by depth is less about a belief in the informedness of a particular heuristic at a point in time and more about hedging, or perhaps more generously, a commitment to explore alternatives to the current supposed best partial solution. We suspect this hedging, coupled with the greedy following of the heuristic for slope steps past the deepest expanded node, accounts for much of Triangle's advantage over GBFS in the $h_{\text{LM-count}}$ setting, where Triangle leads the best enhancement-enabled $h_{\text{LM-count}}$ configuration by 85 instances (Table 2).

## 4 Discussion & Future Work

Triangle search and planning-specific enhancements appear to address many of the same practical failure modes, such as weak heuristic guidance and uninformed heuristic regions. However, they operate differently: triangle search changes the order in which the frontier is expanded, whereas planner enhancements add external guidance signals, queue-control rules, and schedule policies. Standalone triangle search performs well against the base algorithms, but our comparison

to full LAMA shows that it remains weaker than a fully enhanced system overall; on its own, it will not replace such systems without incorporating planning-specific enhancements.

An obvious direction for future work is to add planning-specific enhancements to triangle search. Its strong performance as a base strategy suggests that this combination could yield a stronger planner. That said, the conceptual alignment between specific planning mechanisms and the basic strategy of triangle search may limit the gains from incorporating these enhancements explicitly.

Another possible avenue is to use triangle search as a component of a portfolio planner. The per-domain differences from LAMA (Table 4) indicate that even without incorporating planning-specific enhancements, triangle search would improve coverage on several domains within a common benchmark suite. Questions of parameter tuning, scheduling, and portfolio design would all have to be addressed to make this a practical approach.

Finally, given the large impact of planning-specific enhancements, techniques such as preferred operators and boosting appear broadly useful as dynamic control mechanisms for search. It is currently unclear how to restate them in a way that is meaningful outside of planning's factored state representation and relaxation heuristics. Further, it is unclear if such enhancements would improve overall performance, given the comparatively low overhead of successor generation and heuristic evaluation in many non-planning domains (e.g., permutation puzzles).

## 5 Conclusions

We showed that, in controlled head-to-head comparisons with shared heuristic information and planning-specific enhancements disabled, triangle search consistently outperforms standard graph-search baselines as a base-level strategy. When planning enhancements are re-enabled, triangle search remains competitive with and frequently ahead of single-heuristic enhancement-enabled alternatives. Qualitatively, the domains that benefit most from preferred operators and lazy evaluation overlap with the domains where triangle search captures related behavior implicitly. Our comparisons indicate that this overlap explains part of the advantage of triangle search, and that explicit planning mechanisms still add performance on top.

## Acknowledgements

This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. The computations were enabled by resources provided by the National Academic Infrastructure for Supercomputing in Sweden (NAISS), partially funded by the Swedish Research Council through grant agreement no. 2022-06725. We thank Daniel Gnad for helpful conversations during the development of this paper, and the anonymous reviewers for their feedback, which improved the work.

## References

Aine, S.; Chakrabarti, P.; and Kumar, R. 2007. Awa-a window constrained anytime heuristic search algorithm. In *International Joint Conference on Artificial Intelligence*, 2250–2255.

Bisiani, R. 1987. Beam search. In Shapiro, S. C., ed., *Encyclopedia of Artificial Intelligence*, 56–58. John Wiley and Sons.

Doran, J. E.; and Michie, D. 1966. Experiments with the Graph Traverser program. *Proceedings of the Royal Society A*, 294: 235–259.

Felner, A.; Kraus, S.; and Korf, R. 2003. KBFS: K-best-first search. *Annals of Mathematics and Artificial Intelligence*, 39(1-2): 19–39.

Fickert, M. 2020. A Novel Lookahead Strategy for Delete Relaxation Heuristics in Greedy Best-First Search. In Beck, J. C.; Karpas, E.; and Sohrabi, S., eds., *Proceedings of the Thirtieth International Conference on Automated Planning and Scheduling (ICAPS 2020)*, 119–123. AAAI Press.

Fikes, R. E.; and Nilsson, N. J. 1971. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence*, 2: 189–208.

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

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions of Systems Science and Cybernetics*, SSC-4(2): 100–107.

Helmert, M. 2006. The Fast Downward Planning System. *Journal of Artificial Intelligence Research*, 26: 191–246.

Hoffmann, J.; and Edelkamp, S. 2005. The Deterministic Part of IPC-4: An Overview. *Journal of Artificial Intelligence Research*, 24: 519–579.

Hoffmann, J.; and Nebel, B. 2001. The FF Planning System: Fast Plan Generation Through Heuristic Search. *Journal of Artificial Intelligence Research*, 14: 253–302.

Hoffmann, J.; Porteous, J.; and Sebastia, L. 2004. Ordered Landmarks in Planning. *Journal of Artificial Intelligence Research*, 22: 215–278.

Lemons, S.; Ruml, W.; Holte, R. C.; and Linares López, C. 2024. Rectangle Search: An Anytime Beam Search. In Dy, J.; and Natarajan, S., eds., *Proceedings of the Thirty-Eighth AAAI Conference on Artificial Intelligence (AAAI 2024)*, 20751–20758. AAAI Press.

Lemons, S.; Ruml, W.; López, C. L.; and Holte, R. 2023. Triangle search: An anytime beam search. In *ICAPS 2023 Heuristics and Search for Domain-Independent Planning Workshop*.

Linares López, C.; Celorrio, S. J.; and Olaya, A. G. 2015. The deterministic part of the seventh International Planning Competition. *Artificial Intelligence*, 223: 82–119.

Lipovetzky, N.; and Geffner, H. 2012. Width and Serialization of Classical Planning Problems. In De Raedt, L.; Bessiere, C.; Dubois, D.; Doherty, P.; Frasconi, P.; Heintz,

F.; and Lucas, P., eds., *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI 2012)*, 540–545. IOS Press.

Nakhost, H.; and Müller, M. 2009. Monte-Carlo Exploration for Deterministic Planning. In Boutilier, C., ed., *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*, 1766–1771. AAAI Press.

Pearl, J.; and Kim, J. H. 1982. Studies in Semi-Admissible Heuristics. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-4(4): 391–399.

Pochter, N.; Zohar, A.; Rosenschein, J.; and Felner, A. 2010. Search space reduction using swamp hierarchies. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 24, 155–160.

Pohl, I. 1970. Heuristic search viewed as path finding in a graph. *Artificial Intelligence*, 1: 193–204.

Richter, S.; and Helmert, M. 2009. Preferred Operators and Deferred Evaluation in Satisficing Planning. In Gerevini, A.; Howe, A.; Cesta, A.; and Refanidis, I., eds., *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 2009)*, 273–280. AAAI Press.

Richter, S.; Helmert, M.; and Westphal, M. 2008. Landmarks Revisited. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI 2008)*, 975–982. AAAI Press.

Richter, S.; Thayer, J. T.; and Ruml, W. 2010. The Joy of Forgetting: Faster Anytime Search via Restarting. In Brafman, R.; Geffner, H.; Hoffmann, J.; and Kautz, H., eds., *Proceedings of the Twentieth International Conference on Automated Planning and Scheduling (ICAPS 2010)*, 137–144. AAAI Press.

Richter, S.; and Westphal, M. 2010. The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks. *Journal of Artificial Intelligence Research*, 39: 127–177.

Röger, G.; and Helmert, M. 2010. The More, the Merrier: Combining Heuristic Estimators for Satisficing Planning. In Brafman, R.; Geffner, H.; Hoffmann, J.; and Kautz, H., eds., *Proceedings of the Twentieth International Conference on Automated Planning and Scheduling (ICAPS 2010)*, 246–249. AAAI Press.

Seipp, J.; Pommerening, F.; Sievers, S.; and Helmert, M. 2017. Downward Lab. https://doi.org/10.5281/zenodo.790461.

Taitler, A.; Alford, R.; Espasa, J.; Behnke, G.; Fišer, D.; Gimelfarb, M.; Pommerening, F.; Sanner, S.; Scala, E.; Schreiber, D.; Segovia-Aguas, J.; and Seipp, J. 2024. The 2023 International Planning Competition. *AI Magazine*, 45(2): 280–296.

Torralba, Á.; Seipp, J.; and Sievers, S. 2021. Automatic Instance Generation for Classical Planning. In Goldman, R. P.; Biundo, S.; and Katz, M., eds., *Proceedings of the Thirty-First International Conference on Automated Planning and Scheduling (ICAPS 2021)*, 376–384. AAAI Press.

Vallati, M.; Chrpa, L.; Grześ, M.; McCluskey, T. L.; Roberts, M.; and Sanner, S. 2015. The 2014 International Planning Competition: Progress and Trends. *AI Magazine*, 36(3): 90–98.

Xie, F.; Müller, M.; and Holte, R. C. 2014. Adding Local Exploration to Greedy Best-First Search in Satisficing Planning. In Brodley, C. E.; and Stone, P., eds., *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence (AAAI 2014)*, 2388–2394. AAAI Press.

Xie, F.; Müller, M.; Holte, R. C.; and Imai, T. 2014. Type-based Exploration with Multiple Search Queues for Satisficing Planning. In Brodley, C. E.; and Stone, P., eds., *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence (AAAI 2014)*, 2395–2401. AAAI Press.

Zilberstein, S. 1996. Using Anytime Algorithms in Intelligent Systems. *AI Magazine*, 17(3): 73.