Search Algorithm Portfolios with Multithreaded Components

Takumi Shimoda, Alex Fukunaga

Graduate School of Arts and Sciences
The University of Tokyo
takumi35shimoda@yahoo.co.jp, fukunaga@idea.c.u-tokyo.ac.jp

Abstract

One successful approach to parallel satisficing search is a search algorithm portfolio, where each processor executes an independent search process. Portfolios are competitive with parallel search algorithms where threads use shared Open/-Closed lists. We investigate the scalability of portfolios vs. parallel search algorithm with shared Open/Closed, and show that the benefits due to diversification of portfolio searches hit a plateau, after which adding additional independent components yield diminishing marginal returns. We show that hybrid portfolio/cooperative algorithms where each portfolio component is a multi-threaded, shared Open/Closed search algorithm can significantly outperform both pure portfolios as well as pure shared Open/Closed parallel search algorithms.

1 Introduction

Parallelization is important in order to maximize search algorithm performance on modern, multi-core CPUs. In the case of cost-optimal search, parallelization of the standard A* algorithm (Hart, Nilsson, and Raphael 1968) is somewhat well understood, and practical approaches where multi-threaded approaches which share one or more $Open$/$Closed$ lists have been proposed (Burns et al. 2010; Kishimoto, Fukunaga, and Botea 2013; Phillips, Likhachev, and Koenig 2014; Fukunaga et al. 2017).

On the other hand, parallelization is not well understood for satisficing search where the object is to quickly find any valid solution path (regardless of path cost). Greedy Best First Search (GBFS; Doran and Michie 1966) is a widely used satisficing search algorithm. One straightforward approach to GBFS parallelization is a *cooperative* approach, where all threads share global $Open$ and $Closed$ lists, and all threads expand states from the shared $Open$ according to some best-first criterion. However, the performance of cooperative parallel GBFS is non-monotonic with respect to resource usage – there is a significant risk that using k threads can result in significantly worse performance than using fewer than k threads. It has been shown experimentally that cooperative parallel GBFS can expand orders of magnitude more states than GBFS (Kuroiwa and Fukunaga 2019), and it has been shown theoretically that KPGBFS, a straightforward instance of cooperative parallel GBFS, can expand arbitrarily many more states than GBFS (Kuroiwa and Fukunaga 2020).

A major obstacle to cooperative parallelization of GBFS using shared $Open$/$Closed$ has been that unlike parallel cost-optimal search, there is no obvious set of states which a parallel satisficing search *must* explore in order to be considered a "correct" or "efficient" parallelization of the sequential algorithm. Heusner, Keller, and Helmert (2017) identified the *Bench Transition System* (BTS), the set of all states that can be expanded by GBFS under some tie-breaking policy (conversely, if a state is not in the BTS, there does not exist any tie-breaking strategy for GBFS which will expand that state). Thus, the set of states in the BTS is a natural candidate for an upper bound on the set of states which should be expanded by a parallel GBFS.

OBAT (Shimoda and Fukunaga 2025b) is a recently proposed algorithm that constrains search by (1) expanding only states in the BTS and (2) avoiding simultaneous exploration of multiple benches. This guarantees an upper bound on the number of states expanded relative to sequential GBFS with some tie-breaking policy. However, enforcing these constraints incurs a performance penalty with respect to the state expansion/evaluation rate. Since these algorithms prevent expansion of any state unless it is certain that the state satisfies the expansion constraints, threads can be forced to be idle while they wait until a state which satisfies the expansion constraint becomes available in the shared $Open$.

An alternative approach to parallelization is a *portfolio* (Huberman, Lukose, and Hogg 1997; Gomes and Selman 2001), where multiple *components* independently search the space in parallel. PGBFS is a straightforward portfolio-based approach for parallelizing GBFS which executes k independent components of GBFS, where each component is a 1-thread GBFS instance which uses a different tie-breaking policy and has its own $Open$ and $Closed$ lists. No component can affect the search behavior of any other component. This kind of portfolio can perform well on search spaces where searching multiple regions of the search space can avoid being trapped in some region. Furthermore, if one of the components executes standard sequential GBFS, then it is trivially guaranteed that this approach spends no more

than k times the number of node expansions. However, as k increases, it is possible that diversifying the search in this manner has diminishing returns.

We propose a hybrid portfolio approach for parallelizing GBFS, where each of the portfolio components is a multi-threaded, shared-$Open$/$Closed$ parallel GBFS, e.g., a 24-component portfolio consisting of six, 4-thread OBAT$_S$ components. This combines the strengths of the cooperative parallel algorithms and portfolios, while reducing impact of the scale-related issues with each approach. We show that hybrid portfolios can significantly outperform both cooperative approaches and pure portfolios.

The rest of the paper is structured as follows. Section 2 reviews background and previous work on cooperative and portfolio approaches to GBFS parallelization. Section 3 experimentally evaluates the scalability cooperative and portfolio approaches. Section 4 proposes and evaluates hybrid GBFS portfolios, which use multi-threaded GBFS components within a parallel portfolio. Section 5 concludes with a discussion and directions for future work.

2 Preliminaries and Background

K-Parallel GBFS (KPGBFS) K-Parallel BFS (Vidal, Bordeaux, and Hamadi 2010) is a straightforward, baseline parallelization of BFS. All threads share a single $Open$ and $Closed$. Each thread locks $Open$ to remove a state s with the lowest f-value in $Open$, locks $Closed$ to check duplicates and add $succ(s)$ to $Closed$, and locks $Open$ to add $succ(s)$ to $Open$. KPGBFS is KPBFS with $f(s) = h(s)$.

KPGBFS is an unconstrained, cooperative parallel algorithm. It can expand states which are not in the Bench Transition System, and in general, it can expand arbitrarily more states than sequential GBFS (Kuroiwa and Fukunaga 2020).

One Bench At a Time (OBAT) One Bench At a Time (OBAT) is a constrained parallel algorithm based on KPGBFS. OBAT does not expand any states that would not be expanded by GBFS under any tie-breaking strategy. Furthermore, OBAT guarantees an upper bound on the number of node expansions relative to sequential GBFS (Shimoda and Fukunaga 2025b). OBAT$_S$ is an enhanced version of OBAT with Separate Generation and Evaluation (SGE), which parallelizes both state evaluations and expansions, and improves thread utilization (Shimoda and Fukunaga 2025a).

PGBFS In PGBFS, k independent 1-thread components of GBFS are executed, each with its own $Open$ and $Closed$ lists. Each component expands a node with the best h-value from its local $Open$. Previous work has shown that the search trajectory of best-first search varies significantly depending on tie-breaking (Asai and Fukunaga 2017). Therefore, diversity of search behavior is implemented by using a different policy for breaking ties among states with the same h-value. One component uses FIFO tie-breaking, another component used LIFO tie-breaking, and $k-2$ components use random tie-breaking.

Since 1 component is executing the baseline sequential GBFS, PGBFS trivially guarantees that the parallel portfolio will expand at most k-times the number of states expanded by the baseline sequential GBFS before finding a solution, assuming that all threads execute at the same speed

Shared evaluation cache Although each component of a portfolio search the state independently, a simple way to allow the components to cooperate without affecting the search trajectories of the other components is a shared cache of state evaluations. This eliminates wasted CPU resources due to re-evaluating states which were previously evaluated by other portfolio components.

All portfolios evaluated in this paper use a shared evaluation cache.

3 Scalability of Cooperative Parallel Search and Portfolios

First, we investigate the scalability of cooperative parallel search and portfolios as the number of threads is increased.

We evaluate the performance of KPGBFS, OBAT$_S$, and PGBFS on $k = 8, 12, 24$ threads. As a baseline, we also evaluate sequential GBFS. With $k = 1$ thread, the search behavior (expansion order) of sequential GBFS, KPGBFS, OBAT$_S$, and PGBFS are identical.

We evaluated the algorithms using a set of instances based on the Autoscale-21.11 satisficing benchmark set (42 STRIPS domains, 30 instances/domain, 1260 total instances) (Torralba, Seipp, and Sievers 2021). However, for domains where (1) all methods without GBFS solved all instances (i.e., the instances were too easy), and (2) an instance generator for the domain is available in the Autoscale repository, we replaced the Autoscale-21.11 instances with more difficult instances generated using the Autoscale generator. Specifically, these were agricola, gripper and miconic.

All algorithms use the FF heuristic (Hoffmann and Nebel 2001) as the heuristic evaluation function. Each run had a time limit of 5 min, 2.5GB RAM/thread (e.g., 8 threads: 20GB total) limit on an Intel Xeon CPU E5-2670 v3@2.30GHz processor. KPGBFS and OBAT$_S$ use FIFO tie-breaking. Table 1 shows the coverage (number solved out of 1260), and the geometric means of the number of states expanded until a solution is found, and the state evaluation rates for the 334/1260 instances which were solved by all algorithms evaluated in this paper.

Decreasing Evaluation Rates in Cooperative Parallel Search In order to guarantee that the number of states expanded is bounded relative to sequential GBFS, OBAT$_S$ focuses the search on one specific region of the search space (BTS bench) at a time. Depending on the structure of the search space, enforcing this constraint imposes an inherent limit on the amount of parallelization that can be obtained, as threads must remain idle if they cannot guarantee that they will expand a state within the same bench as the one the search is currently focusing on. Thus, although Tables 1a-1b show that the performance (coverage, expansions) of OBAT$_S$ improves as the number of threads increases, OBAT$_S$ achieves a significantly lower state expansion rate on many instances than KPGBFS and PGBFS, which never require threads to be idle (Table 1c). Furthermore, the gap

#threads	1threads	8 threads	12 threads	24 threads
GBFS	375	-	-	-
KPGBFS	-	477	488	530
OBAT$_S$	-	477	516	536
PGBFS	-	550	564	570

(a) Coverage (number of problems solved out of 1260)

#threads	1threads	8 threads	12 threads	24 threads
GBFS	906	-	-	-
KPGBFS	-	1920	2162	2808
OBAT$_S$	-	1268	1209	1287
PGBFS	-	2732	3787	6850

(b) Number of states expanded (geometric mean)

#threads	1threads	8 threads	12 threads	24 threads
GBFS	5603	-	-	-
KPGBFS	-	32283	44472	75625
OBAT$_S$	-	21835	26501	34393
PGBFS	-	32468	44570	68330

(c) State evaluation rate (states/second, geometric mean)

Table 1: Evaluation on 1260 instances. Means in Tables 1b-1c are for 334 instances solved by all algorithms

between the evaluation rates of OBAT$_S$ vs. KPGBFS and PGBFS increases as the number of threads increases.

Diminishing Marginal Utility with Additional Components in Portfolios In contrast, PGBFS, a pure portfolio algorithm, relies on diversity of search behavior of its components for success, simultaneously exploring multiple regions of the search space (more specifically, as each component in PGBFS is GBFS with a different tie-breaking policy, PGBFS simultaneously explores multiple regions in the Bench Transition System). All of its components are independent, so it achieves high evaluation rates (Table 1c). However, the effect of diversity does not necessarily scale indefinitely as the number of portfolio components increases, and there can be diminishing returns to adding more diversity as the number of threads increases. Although the coverage increases from 377 to 550 as the number of components increases from 1 (GBFS) to 8, the rate of improvement drops rapidly. Increasing from 8 to 12 threads increases coverage by 14, and increasing from 12 to 24 threads only yields an increase of 6 additional problems (564 vs 570) solved.

There are two possible reasons for the diminishing marginal utility of additional portfolio components. First, as components are added, there is an increasing probability that search is being (at least partially) duplicated among the components. Second, depending on the domain, it may be better to focus the search on one (or a few) regions of the BTS instead of searching many regions simultaneously.

Duplicated Search in Portfolios We define the *external hit rate* of the shared cache evaluation cache as the fraction of states evaluated which (according to the cache) were previously evaluated by another portfolio component (excluding the current component). This rate can be used as a proxy for measuring the amount of redundant search being performed by the portfolio components. Figure 1 shows that external hit rates for PGBFS increase significantly as the number of threads increases from 8 to 24 (i.e., search redundancy increases as the number of threads increases).

4 Hybrid Portfolios: Portfolios with Multi-Threaded Components

Cooperative search and portfolios are orthogonal approaches. We therefore propose *hybrid portfolios*, which are portfolios where each component can be a multithreaded, cooperative parallel search algorithm.

Each component in a k-thread hybrid portfolio has fewer threads than a full k-thread cooperative search algorithm, and a k-thread hybrid portfolio has fewer components than a pure k-thread portfolio. Thus, hybrid portfolios seek to obtain the benefits of both cooperative search and portfolios, while avoiding the scaling issues of both approaches by having fewer components and fewer threads per component.

We evaluate two specific classes of hybrid portfolios:

- **POBAT$_S$**: A hybrid portfolio where all components are instances of OBAT$_S$.
- **PKPGBFS**: A hybrid portfolio where all components are instances of KPGBFS.

For both POBAT$_S$ and PKGBFS, one of the portfolio components uses FIFO tie-breaking, another component uses LIFO tie-breaking, and the remaining components use random tie-breaking.

Since every component of POBAT$_S$ is an OBAT$_S$ instance which is constrained to only expand states in the BTS, POBAT$_S$ is guaranteed to only expand states in the BTS, i.e., POBAT$_S$ will only expand a state if sequential GBFS with some tie-breaking will expand it. On the other hand, since KPGBFS is not constrained to search the BTS, PKGBFS search behavior is also similarly unconstrained.

We focus more on POBAT$_S$ because it searches the same Bench Transition System as GBFS and PGBFS, and is more interesting with respect to the question: *"what is the best search strategy to explore the same, constrained search space as GBFS?"*

In general, we could allocate the k total threads in the portfolio arbitrarily, e.g., a 24-thread POBAT$_S$ configuration composed of one 7-thread OBAT$_S$ component, one 13-thread OBAT$_S$ component, and one 4-thread OBAT$_S$ component. In this paper, we only evaluate configurations which allocate k threads among m components, each with the same

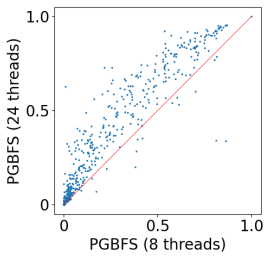


Figure 1: PGBFS External hit rate, $k = 8$ vs. $k = 24$ threads. diagonal lines is $y = x$

#threads	8 threads	12 threads	24 threads
PGBFS	550	564	570
PKPGBFS	542	560	579
POBAT$_S$	540	570	591

(a) Coverage (number of problems solved out of 1260)

#threads	8 threads	12 threads	24 threads
PGBFS	2732	3787	6850
PKPGBFS	1981	2461	4018
POBAT$_S$	1346	1754	3108

(b) Number of states expanded (geometric mean)

#threads	8 threads	12 threads	24 threads
PGBFS	32468	44570	68330
PKPGBFS	31430	43540	75036
POBAT$_S$	26040	36428	56859

(c) State evaluation rate (states/second, geometric mean)

Table 2: Autoscale-21.11/IPC-based benchmark results (1260 instances total). Means in Tables 2b-Table 2c are for 334 instances solved by all algorithms

number of threads. Thus, we denote hybrid portfolio configurations as *AlgorithmName*^{m,n}, where m is the number of components and n is the number of threads per component.

Experimental Evaluation We evaluated the performance of PGBFS , POBAT$_S$ and PKPGBFS with 8, 12, 24 threads. The POBAT$_S$ and PKPGBFS configurations all use 4-thread components, so we evaluated POBAT$_S$ ^{2,4} and PKGBFS^{2,4} on 8 threads, POBAT$_S$ ^{3,4} and PKGBFS^{3,4} on 12 threads, and POBAT$_S$ ^{6,4} and PKGBFS^{6,4} on 24 threads. We used the same experimental setup as in Section 3

Table 2 shows the coverage (number solved out of 1260), and the geometric means of the number of states expanded until a solution is found, and the state evaluation rates for the 334/1260 instances which were solved by all algorithms. Figure 2 shows the number of states expanded, state evaluation rates, and search times for all instances.

Table 2a shows that while the hybrid portfolios and PGBFS have comparable coverage up to $k = 12$ threads, the hybrid portfolios significantly outperform PGBFS on $k = 24$ threads (as well as OBAT$_S$ and KPGBFS from Table 1).

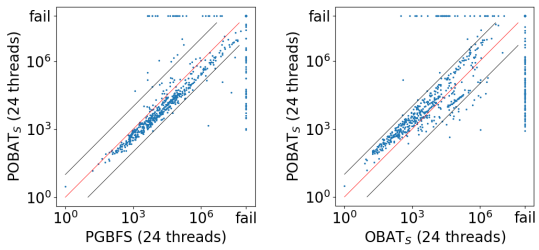
Let ρ =(Δcoverage)/(Δk), the marginal improvement in coverage as the number of threads k increases. When k increases from 8 to 12, $\rho = 18/4$ for PKGBFS and $\rho = 30/4$ for OBAT$_S$. When k increases from 12 to 24, $\rho = 19/12$ for PKGBFS, $\rho = 21/12$ for OBAT$_S$. Thus, as k increases, the performance (coverage) of OBAT$_S$ and PKGBFS scaled better than PGBFS.

Table 2c shows that for all k, POBAT$_S$ has significantly higher evaluation rate than OBAT$_S$.

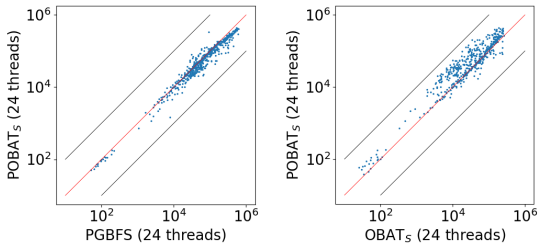
Figure 3 shows that OBAT$_S$ and PKGBFS have significantly lower external hit rates than PGBFS, i.e., these hybrid portfolios perform less redundant search than PGBFS.

5 Conclusion

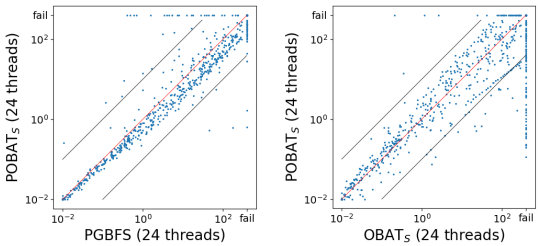
We investigated hybrid portfolios which consist of multi-threaded components where each independent component



(a) Number of states expanded, "fail"= out of time/memory.



(b) State evaluation rates (states/second).



(c) Search time (seconds) "fail"= out of time/memory.

Figure 2: Comparison of OBAT$_S$ vs. PGBFS. Diagonal lines are $y = 0.1x$, $y = x$, and $y = 10x$

is a cooperative algorithm which uses shared *Open*/*Closed* lists. We showed that hybrid portfolios address the scalability issues of both pure portfolios (decreasing marginal utility of additional components) and cooperative algorithms (decreasing state evaluation rates due to idle threads). We limited this study to hybrid portfolios with homogeneous components where all components were the same algorithm with the same number of threads. Future work will investigate hybrid portfolios with heterogeneous components.

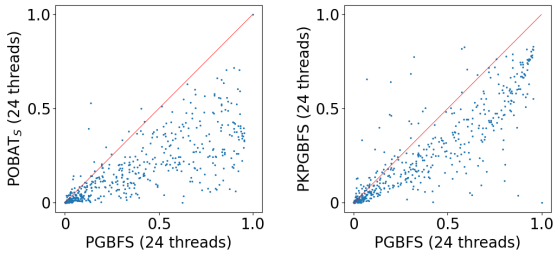


Figure 3: External hit rate. diagonal lines is $y = x$

Acknowledgments

This research was supported by JSPS KAKENHI Grant 24K15083.

References

Asai, M.; and Fukunaga, A. 2017. Tie-Breaking Strategies for Cost-Optimal Best First Search. *J. Artif. Intell. Res.*, 58: 67–121.

Burns, E.; Lemons, S.; Ruml, W.; and Zhou, R. 2010. Best-First Heuristic Search for Multicore Machines. *J. Artif. Intell. Res.*, 39: 689–743.

Doran, J.; and Michie, D. 1966. Experiments with the Graph Traverser Program. In *Proc. Royal Society A: Mathematical, Physical and Engineering Sciences*, volume 294, 235–259.

Fukunaga, A.; Botea, A.; Jinnai, Y.; and Kishimoto, A. 2017. A Survey of Parallel A*. *CoRR*, abs/1708.05296.

Gomes, C. P.; and Selman, B. 2001. Algorithm portfolios. *Artif. Intell.*, 126(1-2): 43–62.

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

Heusner, M.; Keller, T.; and Helmert, M. 2017. Understanding the Search Behaviour of Greedy Best-First Search. In *Proc. SOCS*, 47–55.

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

Huberman, B.; Lukose, R. M.; and Hogg, T. 1997. An Economics Approach to Hard Computational Problems. *Science*, 275(5296): 51–54.

Kishimoto, A.; Fukunaga, A.; and Botea, A. 2013. Evaluation of a simple, scalable, parallel best-first search strategy. *Artif. Intell.*, 195: 222–248.

Kuroiwa, R.; and Fukunaga, A. 2019. On the Pathological Search Behavior of Distributed Greedy Best-First Search. In *Proc. ICAPS*, 255–263.

Kuroiwa, R.; and Fukunaga, A. 2020. Analyzing and Avoiding Pathological Behavior in Parallel Best-First Search. In *Proc. ICAPS*, 175–183.

Phillips, M.; Likhachev, M.; and Koenig, S. 2014. PA*SE: Parallel A* for Slow Expansions. In *Proc. ICAPS*, 208–216.

Shimoda, T.; and Fukunaga, A. 2025a. Decoupling Generation and Evaluation for Parallel Greedy Best-First Search(extended version). In *Proc. SOCS*.

Shimoda, T.; and Fukunaga, A. 2025b. Parallel Greedy Best-First Search with a Bound on Expansions Relative to Sequential Search. In *Proc. AAAI*, 26668–26677.

Torralba, Á.; Seipp, J.; and Sievers, S. 2021. Automatic Instance Generation for Classical Planning. In *Proc. ICAPS*, 376–384.

Vidal, V.; Bordeaux, L.; and Hamadi, Y. 2010. Adaptive K-Parallel Best-First Search: A Simple but Efficient Algorithm for Multi-Core Domain-Independent Planning. In *Proc. SOCS*, 100–107.