

Decoupling Generation and Evaluation for Parallel Greedy Best-First Search

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

In order to understand and control the search behavior of parallel search, recent work has proposed a class of constrained parallel greedy best-first search algorithms which only expands states that satisfy some constraint. However, enforcing such constraints can be costly, as threads must be waiting idly until a state that satisfies the expansion constraint is available. We propose an improvement to constrained parallel search which decouples state generation and state evaluation and significantly improves state evaluation rate, resulting in better search performance.

1 Introduction

Parallelization of combinatorial search algorithms is important in order to maximize search algorithm performance on modern, multi-core CPUs. Greedy Best First Search (GBFS), which repeatedly expands the best state from *Open* according to an evaluation function and a tie-breaking strategy, is a widely used satisficing search algorithm (Doran and Michie 1966). However, the performance of straightforward parallelizations of GBFS is non-monotonic – 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 parallel GBFS can expand orders of magnitude more states than GBFS (Kuroiwa and Fukunaga 2019), and it has been shown theoretically that parallel GBFS using a shared *Open* and/or *Closed* list (including KPGBFS, a straightforward parallelization of GBFS) can expand arbitrarily many more states than GBFS (Kuroiwa and Fukunaga 2020).

Recently, constrained parallel GBFS algorithms which are guaranteed to only expand states which satisfy some *expansion constraint* have been proposed. PUHF (Kuroiwa and Fukunaga 2020) constrains the search so that only states in the *Bench Transition System (BTS)*, the set of all states that can be expanded by GBFS under some tie-breaking order (Heusner, Keller, and Helmert 2017) are expanded. OBAT (Shimoda and Fukunaga 2025) further constrains the search so that multiple benches are not simultaneously explored.

However, such constraints incur a cost, as threads can be forced to be idle while waiting for a state which is guaranteed to satisfy the expansion constraint becomes available. Due to this idle waiting (poor CPU utilization), it has been shown that PUHF and OBAT both have a significantly lower state expansion rate compared to unconstrained parallel GBFS. As a result, even on problems where constrained parallel GBFS finds a solution with fewer expansions than unconstrained parallel GBFS, constrained parallel GBFS can underperform unconstrained parallel GBFS.

We propose Separate Generation and Evaluation (SGE), which decouples state expansion and evaluation so that instead of waiting for a single thread to fully expand a state (generating and evaluating its successors), multiple threads evaluate the successors. We show that this significantly improves the state evaluation rate in constrained parallel GBFS, resulting in significantly improved performance.

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 $\text{succ}(s)$ to *Closed*, and locks *Open* to add $\text{succ}(s)$ to *Open*. KPGBFS is KPBF with $f(s) = h(s)$.

Constrained Parallel GBFS Recent work has investigated parallel GBFS algorithms based on KPGBFS which expand states only if they satisfy some expansion constraint. Parallel Under High-water mark First (PUHF) and its successors PUHF2–4 (Kuroiwa and Fukunaga 2020; Shimoda and Fukunaga 2023) expanded only states which are guaranteed to be in the *BTS* (Heusner, Keller, and Helmert 2017), but unlike sequential GBFS, PUHF can simultaneously expand states in multiple benches. One Bench At a Time (OBAT) further constrains the search to expand states in a single bench at a time, guaranteeing that the number of states expanded is bounded relative to sequential GBFS with some time-breaking order (Shimoda and Fukunaga 2025).

Constrained Parallel GBFS (CPGBFS) (Algorithm 1) is a schema for a class of parallel search algorithms based on KPGBFS, which only expands nodes which satisfy some

Algorithm 1: CPGBFS: Constrained Parallel GBFS

```
1:  $Open \leftarrow \{s_{init}\}$ ,  $Closed \leftarrow \{s_{init}\}$ ;  $\forall i, s_i \leftarrow NULL$ 
2: for  $i \leftarrow 0, \dots, k-1$  in parallel do
3:   loop
4:     with  $lock(Open)$ 
5:       if  $Open = \emptyset$  then
6:         if  $\forall j, s_j = NULL$  then
7:            $unlock(Open)$ ; return  $NULL$ 
8:         else if  $satisfies(top(Open)) = true$  then
9:            $s_i \leftarrow top(Open)$ ;  $Open \leftarrow Open \setminus \{s_i\}$ 
10:        if  $s_i = NULL$  then continue
11:        if  $s_i \in s_{goal}$  then return  $Path(s_i)$ 
12:        for  $s'_i \in succ(s_i)$  do
13:           $lock(Closed)$ 
14:          if  $s'_i \notin Closed$  then
15:             $Closed \leftarrow Closed \cup \{s'_i\}$ 
16:             $unlock(Closed)$ 
17:             $children(s_i) \leftarrow children(s_i) \cup \{s'_i\}$ 
18:             $evaluate(s'_i)$ 
19:          else
20:             $unlock(Closed)$ 
21:          with  $lock(Open)$ 
22:            for  $s'_i \in children(s_i)$  do
23:               $Open \leftarrow Open \cup \{s'_i\}$ 
24:           $s_i \leftarrow NULL$ 
```

algorithm-specific constraint in line 8, where *satisfies* is a function which returns *true* if and only if *s* satisfies the algorithm-specific expansion constraint.

KPGBFS is a special case of CPGBFS where *satisfies* always returns *true*. The previously proposed constrained search algorithms (PUHF, PUHF2–4, and OBAT) are instances of CPGBFS where the *satisfies* function implements a check for the sufficient constraint which guarantees that *s* is in the BTS (and in the case of OBAT, further constrains the expansion to prevent simultaneous expansion in multiple benches) – the specific implementation details of *satisfies* depend on the specific algorithm.¹

3 State Expansion Bottlenecks in Constrained Parallel Search

All previous CPGBFS variants (all PUHF variants and OBAT) have a significantly lower state evaluation rate than unconstrained parallel search (KPGBFS). There are two related reasons: (1) the expansion constraint, and (2) batch successor insertion.

Expansion Constraint Bottleneck Unconstrained parallel search algorithms such as KPGBFS will unconditionally expand the top states in *Open*. Threads in unconstrained algorithms are only idle when waiting for a mutex lock for the shared *Open* and *Closed* structures, and such mutex overheads can be greatly reduced by efficiently implementing *Open/Closed* (e.g., sharded implementations).

¹The original presentations of PUHF presented these algorithms as marking states guaranteed to be in the BTS as *certain*, and only expanding nodes marked as *certain*, but it is straightforward to reframe this as a constraint check as in Algorithm 1.

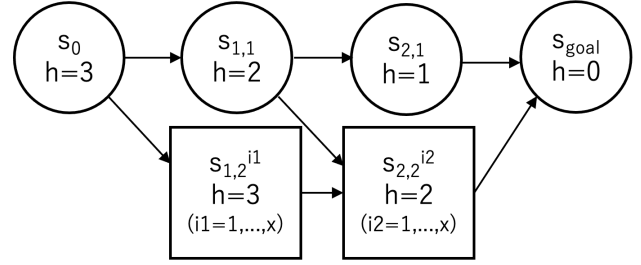


Figure 1: Example of parallel expansion bottleneck

In contrast, CPGBFS cannot expand the top state $top(Open)$ unless the expansion constraint is satisfied, even if the mutex lock is available. For example, in Figure 1 the only states in the *BTS* are the circular nodes $(s_0, s_{1,1}, s_{2,1}, s_{goal})$, so any algorithm which constrains expansions to states in the *BTS* can only expand one of these states at a time, while all other threads must wait.

Batch Successor Insertion Bottleneck Consider Figure 1. In a standard, single-thread implementation of best-first search with eager evaluation, the expansion of $s_{1,1}$ includes (1) generating $succ(s_{1,1}) = s_{2,1}, s_{2,2}^1, \dots, s_{2,2}^x$, (2) evaluating all states in $succ(s_{1,1})$ with a heuristic evaluation function, and (3) inserting $succ(s_{1,1})$ in *Open*. In many cases, computing the heuristic evaluation function consumes the majority of time spent expanding the state, and the full expansion of a single state such as $s_{1,1}$ can take a significant amount of time due to the evaluation of all of its successors.

A constrained parallel search which seeks to expand a similar set of nodes as GBFS has an additional requirement not present in single-threaded search: the successors of a state *s* are all simultaneously inserted in *Open* only after all successors of state *s* are evaluated (Algorithm 1, lines 21–23). This batch insertion ensures that the successors of *s* are expanded in best-first order – otherwise (e.g., if states were inserted one at a time directly into *Open* immediately after being evaluated), a state s' with a worse *f*-value than its sibling s'' might be inserted into *Open* and expanded by another thread before s'' has been inserted into *Open*.

The state evaluation rate of unconstrained parallel search (e.g., KPGBFS), is not significantly affected by whether the successors are inserted in a single batch or one at a time, because available threads can freely expand the top states from *Open*. However, in CPGBFS, the combination/interaction of the expansion constraint and the batch successor insertion requirement causes a significant bottleneck. For example, in Figure 1, in Algorithm 1, if a thread starts to expand $s_{1,1}$, all other threads must stop and wait until all of $succ(s_{1,1})$ have been fully evaluated and inserted into *Open*.

4 Separate Generation and Evaluation

We propose Separate Generation and Evaluation (SGE), an approach for increasing the state evaluation rate in CPGBFS. SGE alleviates the batch successor insertion bottleneck described above.

Algorithm 2: Constrained Parallel GBFS with SGE

```
1:  $Open \leftarrow \{s_{init}\}, Closed \leftarrow \{s_{init}\}; \forall i, s_i \leftarrow NULL$ 
2: for  $i \leftarrow 0, \dots, k - 1$  in parallel do
3:   loop
4:      $lock(Unevaluated)$ 
5:     if  $Unevaluated \neq \emptyset$  then
6:        $s_i \leftarrow top(Unevaluated)$ 
7:        $Unevaluated \leftarrow Unevaluated \setminus \{s_i\}$ 
8:        $unlock(Unevaluated)$ 
9:        $evaluate(s_i)$   $\triangleright$  cached evaluation
10:      with  $lock(Evaluated\ Successor)$ 
11:         $Evaluated\ Successor(\text{parent of } \{s_i\}) \leftarrow$ 
12:         $Evaluated\ Successor(\text{parent of } \{s_i\}) \cup \{s_i\}$ 
13:      if all siblings of  $s_i$  have been evaluated then
14:        with  $lock(Open), lock(Closed)$ 
15:          for  $s'_i \in \text{siblings of } s_i$  do
16:            if  $s'_i \notin Closed$  then
17:               $Closed \leftarrow Closed \cup \{s'_i\}$ 
18:               $Open \leftarrow Open \cup \{s'_i\}$ 
19:        else
20:           $unlock(Unevaluated)$ 
21:          with  $lock(Open)$ 
22:            if  $Open = \emptyset$  then
23:               $unlock(Open)$ 
24:              if  $\forall j, s_j = NULL$  then
25:                return  $NULL$ 
26:            else if  $satisfies(top(Open)) = true$  then
27:               $s_i \leftarrow top(Open); Open \leftarrow Open \setminus \{s_i\}$ 
28:            if  $s_i = NULL$  then continue
29:            if  $s_i \in s_{goal}$  then return  $Path(s_i)$ 
30:          with  $lock(Unevaluated)$ 
31:             $Unevaluated \leftarrow Unevaluated \cup succ(s'_i)$ 
32:           $s_i \leftarrow NULL$ 
```

Continuing the Figure 1 example from the previous section, instead of waiting idly while one thread expands $s_{1,1}$ (which includes computing all of the heuristic values for $s_{2,1}, s_{2,2}, \dots, s_{2,2}^x$), it would be more efficient to parallelize the evaluation of $s_{2,1}, s_{2,2}^1, \dots, s_{2,2}^x$ among available threads.

The main idea of SGE is to decompose the expansion of state s into separate units of work which can be parallelized: (1) successor generation, which generates $succ(s)$, the successors of s , and (2) successor evaluation, which evaluates $succ(s)$. Algorithm 2 shows Constrained Parallel GBFS with SGE. After a thread selects s for expansion from the shared $Open$, it generates $succ(s)$, and inserts $succ(s)$ into the shared $Unevaluated$ queue. The evaluation of states in $Unevaluated$ is done in parallel, taking precedence over selection of states for expansion (a thread will select a state for expansion from $Open$ only if $Unevaluated$ is currently empty (Algorithm 2, line 5)).

Evaluated states are *not* immediately inserted into $Open$. Instead, we insert all of the successors of s simultaneously into $Open$, after they have all been evaluated (lines 13–18). This is so that the parallel search is able to prioritize $succ(s)$ similarly to GBFS.

Consider the behavior of PUHF (which only expands states in the BTS) with SGE on the search space in Figure 1.

First, a thread pops s_0 from $Open$ ($satisfies(s_0) = true$), and generates its successors ($s_{1,1}, s_{1,2}^1, \dots, s_{1,2}^x$), which are inserted in $Unevaluated$. Available threads will pop these successors from $Unevaluated$ and evaluate them. When all successors of s_0 have been evaluated, they are all inserted in $Open$. Next, some thread removes $s_{1,1}$ ($satisfies(s_{1,1}) = true$), generates its successors ($s_{2,1}, s_{2,2}^1, \dots, s_{2,2}^x$), and inserts them in $Unevaluated$. While generating the successors of $s_{1,1}$, other threads may try to pop a state from $Open$, but since the top state at that time ($s_{1,2}^i$) is not in the BTS ($satisfies(s_{1,2}^i) = false$), $Open$ will remain untouched. After the successors of $s_{1,1}$ have been inserted in $Unevaluated$, the available threads will remove them from $Unevaluated$ and evaluate them. The search continues until s_{goal} is found. Each time a state’s successors are generated, available threads evaluate the successors in parallel. This clearly improves thread utilization compared to CPG-BFS without SGE, where only 1 thread is active throughout the search space in Figure 1.

The basic idea of decoupling state generation and state evaluation is similar to that of ePA*SE, which decouples state generation and *edge evaluations* in a parallel A* (Mukherjee, Aine, and Likhachev 2022), where an edge evaluation is the computation required to evaluate the application of an operator, (e.g., collision checking using a simulation model in robot motion planning). Because the requirements and objectives of GBFS (satisficing search) and A* (cost-optimal search) differ, the implementation of SGE is somewhat simpler (using an $Unevaluated$ queue instead of dummy/real edges as in ePA*SE).

SGE and search behavior The state expansion order of a parallel search algorithm A with SGE will differ from the expansion order of A without SGE. Although it is nontrivial to precisely characterize the difference between the expansion order of an algorithm with and without SGE, a simple approximation is that executing a parallel search algorithm A with SGE on k threads is somewhat similar to executing A without SGE on $m < k$ threads, where each of the m “threads” is faster than each of the actual k threads.

As a simple example, consider searching a state space which is a tree with uniform branching factor 2, where the heuristic evaluation function computation is the computational bottleneck, and assume that $Open$ currently contains many nodes. With $k = 16$ threads, KPGBFS will expand 16 states at a time – each thread expands 1 state, where the expansion includes generation and heuristic evaluation of the state’s 2 successors. In contrast, KPGBFS with SGE would be evaluating 16 states at a time – each thread, after quickly generating the successors of a state, would then be assigned to evaluate 1 successor state, essentially the same as KPG-BFS without SGE expanding 8 states, i.e., similar to KPG-BFS without SGE running on 8 threads.

The efficiency (number of states expanded) of parallel GBFS compared to sequential GBFS tends to worsen as the number of threads increases, so the tendency of SGE to cause the parallel search to behave as if there were fewer threads can result in more efficient search (fewer state expansions), compared to parallel search without SGE.

SGE and expansion constraints The state expansion constraints (i.e., the *satisfies* check) for the various CPG-BFS algorithms known to date are defined based on: (a) comparisons between the h -value of a state’s parent and the h -values of the siblings of s (for PUHF), or (b) $h(s)$ vs. the h -values of other states currently being expanded (for PUHF2–4, OBAT). Therefore, distributing the evaluation of the successors of s among multiple threads has no impact on the correctness of the expansion constraint (i.e., the guarantee that the node being expanded is in the BTS).

5 Experimental Evaluation of SGE

We evaluated SGE using the planning benchmark set used to evaluate parallel GBFS variants in (Shimoda and Fukunaga 2025). These are based on the Autoscale-21.11 benchmark set (42 STRIPS domains, 30 instances/domain, 1260 total instances) (Torralba, Seipp, and Sievers 2021), except that gripper and miconic were replaced with harder instances because the original Autoscale instances were too easy to distinguish among the parallel algorithms. All search algorithms use the FF heuristic (Hoffmann and Nebel 2001). Each run has a run time limit of 5 minutes and 3 GB RAM/thread (e.g., 24 GB total for a 8-thread run) limit on a Intel(R) Xeon(R) CPU E5-2670 v3 @ 2.30GHz processor.

We evaluated KPGBFS, KPGBFS with SGE (KPGBFS_S), PUHF3, PUHF3 with SGE (PUHF3_S), OBAT, and OBAT with SGE (OBAT_S) on $k \in \{4, 8, 16\}$ threads. We also report baseline single-threaded GBFS results. All tie-breaking is First-In-First-Out. The code is available at <https://github.com/TakuShimoda/AAAI25>.

Table 1a compares the state evaluation rates of the algorithms. Table 1b shows the number of instances solved by each algorithm. In Table 1a, as well as % improvements mentioned below, we include only the 354 instances solved by all algorithms so that means can be computed. Scatterplots of evaluation rate, number of states expanded, and search time are in (Shimoda and Fukunaga 2024).

Evaluation Rates Table 1a shows that the constrained algorithms (PUHF3, OBAT) have a significantly lower evaluation rate (states/second) than the unconstrained KPGBFS. The evaluation rate of KPGBFS_S is somewhat lower than KPGBFS for $k \in \{4, 8, 16\}$ threads. Thus, management of the overhead of a separate *Unevaluated* queue in SGE imposes a noticeable evaluation rate penalty for an unconstrained search. The effect of this overhead is noticeable for the problems with the highest evaluation rate (Shimoda and Fukunaga 2024). On the other hand, PUHF3_S and OBAT_S have significantly higher evaluation rates than PUHF3 and OBAT, respectively, showing that SGE successfully achieves the goal of improving the evaluation rate for constrained parallel best-first search.

Number of States Expanded For 16 threads, KPGBFS_S, PUHF3_S, OBAT_S expanded 17.4%, 14.8%, 10.5% fewer states than KPGBFS, PUHF3, OBAT, respectively. Thus, SGE has the effect of reducing the search required to solve problem instances for both constrained and unconstrained parallel GBFS. See scatterplots in (Shimoda and Fukunaga 2024) for details.

#threads	1 thread	4 threads	8 threads	16 threads
GBFS	4814	-		
KPGBFS	-	15271	27521	48772
KPGBFS _S	-	15030	26309	45668
PUHF3	-	11804	16806	22121
PUHF3 _S	-	13295	21156	31814
OBAT	-	10221	12830	15659
OBAT _S	-	12407	18713	24634

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

#threads	1 thread	4 threads	8 threads	16 threads
GBFS	401	-		
KPGBFS	-	462	488	529
KPGBFS _S	-	472	500	532
PUHF3	-	459	477	494
PUHF3 _S	-	468	494	510
OBAT	-	458	477	496
OBAT _S	-	478	506	532

(b) Coverage results (out of 1260 total instances)

Table 1: Autoscale-21.11/IPC-based benchmark results (1260 instances total). Means in Table 1a are for 354 instances solved by all algorithms

Search Time The differences in state evaluation rate and search efficiency result in significantly improved search times overall. For 16 threads, KPGBFS_S, PUHF3_S, OBAT_S, had 14.1%, 39.1%, 38.5% faster search time than KPGBFS, PUHF3 and OBAT, respectively. For 8 threads, KPGBFS_S, PUHF3_S, OBAT_S, had 12.4%, 27.5%, 27.8% faster search time than KPGBFS, PUHF3 and OBAT, respectively. For 4 threads, KPGBFS_S, PUHF3_S, OBAT_S, had 11.2%, 19.7%, 17.6% faster search time than KPGBFS, PUHF3 and OBAT, respectively. See scatterplots in (Shimoda and Fukunaga 2024) for details.

Coverage Table 1b shows that SGE significantly improves the number of instances solved by PUHF3 and OBAT.

6 Discussion and Conclusion

We proposed SGE, an approach to increase state evaluation rates in constrained parallel search algorithm by separating successor generation and evaluation. We showed SGE significantly increases the state evaluation rate of PUHF3 and OBAT, resulting in significantly improved overall performance and coverage. The batch successor insertion state bottleneck addressed by SGE (Section 3) arises when implementing a parallel search algorithm which seeks to behave similarly to single-threaded GBFS with a standard eager evaluation policy, where states are evaluated immediately after they are generated and before being inserted into *Open*. In lazy (deferred) evaluation (Richter and Helmert 2009), where states are not evaluated before insertion into *Open* and are inserted into *Open* based on their parent’s f -value (and later evaluated when they are expanded), the batch successor insertion bottleneck would not apply. Search with lazy evaluation behaves quite differently than search with eager evaluation, and parallel satisficing search with lazy evaluation is an avenue for future work.

Acknowledgments

This research was supported by JSPS KAKENHI Grant 20K11932 and JST SPRING Grant 250800000426.

References

- 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.
- 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.
- 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.
- Mukherjee, S.; Aine, S.; and Likhachev, M. 2022. ePA*SE: Edge-Based Parallel A* for Slow Evaluations. In Chrupa, L.; and Saetti, A., eds., *Proc. SOCS*, 136–144.
- Richter, S.; and Helmert, M. 2009. Preferred Operators and Deferred Evaluation in Satisficing Planning. In *Proc. ICAPS*, volume 19, 273–280.
- Shimoda, T.; and Fukunaga, A. 2023. Improved Exploration of the Bench Transition System in Parallel Greedy Best First Search. In *Proc. SOCS*, 74–82.
- Shimoda, T.; and Fukunaga, A. 2024. Decoupling Generation and Evaluation for Parallel Greedy Best-First Search(extended version). *CoRR*, abs/2408.05682.
- Shimoda, T.; and Fukunaga, A. 2025. 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.