Parallel Greedy Best-First Search with a Bound on Expansions Relative to Sequential 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

Parallelization of non-admissible search algorithms such as GBFS poses a challenge because straightforward parallelization can result in search behavior which significantly deviates from sequential search. Previous work proposed PUHF, a parallel search algorithm which is constrained to only expand states that can be expanded by some tie-breaking strategy for GBFS. We show that despite this constraint, the number of states expanded by PUHF is not bounded by a constant multiple of the number of states expanded by sequential GBFS with the worst-case tie-breaking strategy. We propose and experimentally evaluate One Bench At a Time (OBAT), a parallel greedy search which guarantees that the number of states expanded is within a constant factor of the number of states expanded by sequential GBFS with some tie-breaking policy.

1 Introduction

Best-first search (BFS) algorithms such as A^*(Hart, Nilsson, and Raphael 1968) and GBFS (Doran and Michie 1966) are a key component of planning and scheduling systems. In order to fully exploit the increasing number of cores on modern CPUs, it is necessary to parallelize these BFS algorithms. Successful approaches to parallelization have been developed for admissible BFS algorithms such as A^*(Burns et al. 2010; Kishimoto, Fukunaga, and Botea 2013; Phillips, Likhachev, and Koenig 2014; Fukunaga et al. 2017).

Parallelization of GBFS has turned out to be more challenging. Parallel portfolios which execute completely independent instances of GBFS (with a different tie-breaking on each thread), e.g., (Kuroiwa and Fukunaga 2020) are a simple approach. However, this approach does not exploit the potential for explicit cooperation among threads.

A natural approach to implement a more "cooperative" multi-threaded search is to use shared data structures ($Open$, $Closed$), similar to parallel A^*. It is easy to implement a state expansion *policy* which seems similar to sequential GBFS, e.g., "expand a state with the best heuristic value in the (shared) $Open$ list". However the resulting state expansion *behavior* of such a parallel GBFS can deviate drastically from that of sequential GBFS. This poses two issues: (1) the relationship between the states expanded by parallel vs. sequential GBFS is nontrivial to analyze theoretically, and it is difficult to answer basic questions such as: "what is the worst-case performance of this parallel GBFS relative to sequential GBFS?" (2) For some problems, the empirical performance of parallel GBFS is much worse than sequential GBFS, resulting in orders of magnitude of slowdown on some IPC benchmarks (Kuroiwa and Fukunaga 2019), and in general, the number of states expanded by simple approaches to parallel GBFS with shared $Open$ and/or $Closed$ lists is not bounded by a constant factor relative to sequential GBFS (Kuroiwa and Fukunaga 2020).

Previous work proposed PUHF (Kuroiwa and Fukunaga 2020), a parallel GBFS algorithm which is constrained to only expand states in the Bench Transition System (BTS), the set of states that can be expanded by some tie-breaking strategy for GBFS (Heusner, Keller, and Helmert 2017). The BTS provides a natural theoretical constraint on the states expanded by an algorithm which "behaves similarly to sequential GBFS", somewhat analogous to the (much stronger) constraint that a parallel A^*must expand all states with evaluation value less than the optimal. However, although PUHF guarantees that only states in the BTS are expanded, it does not guarantee a bound on the number of states expanded compared to sequential GBFS, i.e., PUHF provides a qualitative guarantee about search behavior, but it does not provide any performance guarantee.

Thus, previous parallel GBFS algorithms with shared $Open$/$Closed$ lists pose an *unbounded risk* – on any given instance, parallel GBFS may outperform sequential GBFS, or it may perform much worse, with no known general bound on how badly search performance can degrade. In this paper, we directly address the problem of establishing a bound on the number of states expanded by parallel GBFS relative to sequential GBFS. While PUHF constrained expansion to only states which could be expanded by sequential GBFS with some tie-breaking strategy, we propose a new algorithm which further constrains the search so that it explores only one bench at a time. This allows us to bound the number of state expansions by parallel search to be no worse than the number of states expanded by sequential search with some tie-breaking policy plus a small constant.

The main contributions of this paper are: (1) We propose One Bench At a Time (OBAT), the first parallel search algorithm with shared *Open*/*Closed* lists with a guaranteed bound on the number of expansions relative to sequential GBFS. (2) We show that this theoretical guarantee does not require sacrificing practical performance – OBAT, combined with SGE (a method that parallelizes both state generation and evaluation) (Shimoda and Fukunaga 2024b), performs comparably to baselines which have no such guarantee.

The rest of the paper is structured as follows. After a review of preliminaries and background (Sec. 2), we first show that the number of states expanded by PUHF is not bounded by a constant multiple of the number of states expanded by GBFS with the worst-case tie-breaking strategy (Sec. 3). We then propose One Bench At a Time (OBAT), which guarantees that the number of states expanded is within a constant factor of the number expanded by sequential GBFS with some tie-breaking policy (Sec. 4-5). We also propose OBAT$_S$, which significantly improves the state evaluation rate by using SGE (Sec. 6). We experimentally show that although the number of states expanded by OBAT is competitive with previous parallel algorithms, the state evaluation rate is significantly lower. However, we show that OBAT$_S$ is competitive with previous methods (Sec. 7). An extended version of this paper which includes the Supplement (additional technical details) is available (Shimoda and Fukunaga 2024a).

2 Preliminaries and Background

State Space Topology State space topologies are defined following Heusner, Keller, and Helmert (2018).

Definition 1. A *state space* is a 4-tuple $\mathcal{S} = \langle S, succ, s_{init}, S_{goal} \rangle$, where S is a finite set of states, $succ : S \rightarrow 2^S$ is the successor function, $s_{init} \in S$ is the initial state, and $S_{goal} \subseteq S$ is the set of goal states. If $s' \in succ(s)$, we say that s' is a successor of s and that $s \rightarrow s'$ is a (state) transition. $\forall s \in S_{goal}, succ(s) = \emptyset$. A *heuristic* for $\mathcal{S}$ is a function $h : S \rightarrow \mathbb{R}$ and $\forall s \in S_{goal}, h(s) = 0$. A *state space topology* is a pair $\langle \mathcal{S}, h \rangle$, where $\mathcal{S}$ is a state space.

We call a sequence of states $\langle s_0, ..., s_n \rangle$ a *path* from s_0 to s_n, and denote the set of paths from s to s' as $P(s, s')$. s_i is the i th state in a path p and $|p|$ is the length of p. A *solution* of a state space topology is a path p from s_{init} to a goal state. We assume at least one goal state is reachable from s_{init}, and $\forall s \in S, s \notin succ(s)$.

Best-First Search Best-First Search (BFS) is a class of search algorithms that use an evaluation function $f : S \rightarrow \mathbb{R}$ and a tie-breaking strategy τ. BFS searches states in the order of evaluation function values (f-values). States with the same f-value are prioritized by τ. In Greedy Best-First Search (GBFS; Doran and Michie 1966), $f(s) = h(s)$.

K-Parallel GBFS (KPGBFS) K-Parallel BFS (Vidal, Bordeaux, and Hamadi 2010) is a straightforward, baseline parallelization of BFS. All k 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 $f(s) = h(s)$. Algorithm 1 shows the pseudocode for KPGBFS.

Algorithm 1: K-Parallel GBFS (KPGBFS)

```
1:  Open ← {s_init}, Closed ← {s_init}; ∀i, s_i ← NULL
2:  for i ← 0, ..., k − 1 in parallel do  ▷ k is the number of threads
3:      loop
4:          while s_i = NULL do
5:              lock(Open)
6:              if ∀j, s_j = NULL then
7:                  if Open = ∅ then
8:                      unlock(Open); return NULL
9:              else
10:                 s_i ← top(Open); Open ← Open \ {s_i}
11:             unlock(Open)
12:         if s_i ∈ S_goal then return s_i
13:         lock(Open), lock(Closed)
14:         for s'_i ∈ succ(s_i) do
15:             if s'_i ∉ Closed then
16:                 Closed ← Closed ∪ {s'_i}
17:                 Open ← Open ∪ {s'_i};
18:         unlock(Open), unlock(Closed)
19:         s_i ← NULL
```

T-bounded and T-pathological Kuroiwa and Fukunaga (2020) proposed the following in order to classify the quantitative behavior of parallel BSF algorithms.

Definition 2. Given a state space topology $\mathcal{T}$, a search algorithm A is *t-bounded* relative to search algorithm B on $\mathcal{T}$ iff A performs no more than t-times as many expansions as B. A is *t-pathological* relative to search algorithm B on $\mathcal{T}$ iff A is not t-bounded relative to B on $\mathcal{T}$.

A is t-bounded relative to B iff A is t-bounded relative to B for all state space topologies. A is t-bounded relative to B on state space topologies with the property P iff A is t-bounded relative to B for all state space topologies with P.

A is *pathological* relative to B iff for all $t > 0$ there exists a state space topology $\mathcal{T}$, such that A is t-pathological relative to B on $\mathcal{T}$. A is pathological relative to B on state space topologies with property P iff for all $t > 0$ there exists a state space topology $\mathcal{T}$ with property P such that A is t-pathological relative to B on $\mathcal{T}$.

Kuroiwa and Fukunaga (2020) showed that straightforward parallelizations of GBFS with shared *Open* and/or *Closed*, including KPGBFS, are pathological.

Bench Transition Systems Heusner et al. (2017) defined progress states and bench transition systems in order to characterize the behavior of GBFS, building upon the definition of high-water marks by Wilt and Ruml (2014).

Definition 3. Let $\langle \mathcal{S}, h \rangle$ be a state space topology with states S and $P(s) = \{p \in P(s, s') \mid s' \in S_{goal}\}$. The high-water mark of $s \in S$ is

$$hwm(s) := \begin{cases} \min_{p \in P(s)}(\max_{s' \in p} h(s')) & \text{if } P(s) \neq \emptyset \\ \infty & \text{otherwise} \end{cases}$$

The high-water mark of a set of states $S' \subseteq S$ is defined as

$$hwm(S') := \min_{s \in S'} hwm(s)$$

Definition 4. A state s of a state space topology $\langle \mathcal{S}, h \rangle$ is a *progress state* iff $hwm(s) > hwm(succ(s))$.

Definition 5. Let $\langle \mathcal{S}, h \rangle$ be a state space topology with a set of states S. Let $s \in S$ be a progress state.

The *bench level* of s is $level(s) = hwm(succ(s))$.

The *inner bench states* $inner(s)$ for s consist of all states $s'' \neq s$ that can be reached from s on paths on which all states $s' \neq s$ (including s'' itself) are non-progress states and satisfy $h(s') \leq level(s)$.

The *bench exit states* $exit(s)$ for s consist of all progress states s' with $h(s') = level(s)$ that are successors of s or of some inner bench state of s.

The *bench states* $states(s)$ for s are $\{s\} \cup inner(s) \cup exit(s)$.

The *bench induced by* s, denoted by $\mathcal{B}(s)$, is the state space with states $states(s)$, initial state s, and goal states $exit(s)$. The successor function is the successor function of S restricted to $states(s)$ without transitions to s and from bench exit states $exit(s)$.

Definition 6. Let $\mathcal{T} = \langle \mathcal{S}, h \rangle$ be a state space topology with initial state s_{init}. The bench transition system $\mathcal{B}(\mathcal{T})$ of $\mathcal{T}$ is a directed graph $\langle V, E \rangle$ whose vertices are benches. The vertex set V and directed edges E are inductively defined as the smallest sets that satisfy the following properties:

1. $\mathcal{B}(s_{init}) \in V$
2. if $\mathcal{B}(s) \in V$, $s' \in exit(s)$, and s' is a non-goal state, then $\mathcal{B}(s') \in V$ and $\langle \mathcal{B}(s), \mathcal{B}(s') \rangle \in E$

Theorem 1. (Heusner, Keller, and Helmert 2017) Let $\mathcal{T} = \langle \mathcal{S}, h \rangle$ be a state space topology with set of states S and bench transition system $\langle V, E \rangle$. For each state $s \in S$, it holds that $s \in \mathcal{B}(s')$ for some $\mathcal{B}(s') \in V$ iff there is a tie-breaking strategy with which GBFS expands s.

BTS-Constrained Search By Theorem 1, the bench transition system (BTS) defines the set of all states which are candidates for expansion by GBFS with some tie-breaking strategy.

Restricting the search to only expand states which are in the BTS is a natural constraint for parallel GBFS.

Definition 7. A search algorithm is BTS*-constrained* if it expands only states which are in the BTS (Shimoda and Fukunaga 2023).

PUHF: A BTS-Constrained Parallel GBFS Kuroiwa and Fukunaga (2020) proposed Parallel Under High-water mark First (PUHF), a BTS-constrained parallel GBFS. PUHF marks states which are guaranteed to be in the BTS as *certain*, and only expands states marked as certain. The criterion used by PUHF to mark states as certain was a restrictive, sufficient (but not necessary) condition for being in the BTS. Recently, looser sufficient conditions for marking states were proposed, resulting in PUHF2–4, which significantly improved performance over PUHF (Shimoda and Fukunaga 2023).

Constrained vs. Unconstrained Parallel GBFS More generally, we say that a best-first search algorithm is *constrained* if it expands the best state s in *OPEN* only if s satisfies some additional constraint C. For example, PUHF is constrained, as it expands the best state in *OPEN* only if it is in the BTS. We say that a best-first search algorithm is *unconstrained* if it unconditionally expands the best state s in *OPEN*. For example, KPGBFS is unconstrained.

SGE: Separate Generation and Evaluation Unconstrained parallel algorithms such as KPGBFS will unconditionally expand the top states in $Open$, so threads will be kept fully busy as long as $Open$ is not empty. In contrast, constrained algorithms such as PUHF can only expand states when it is guaranteed that they satisfy some state expansion constraint. Even if $Open$ is non-empty, threads will remain idle until a state which satisfies the expansion constraint becomes available. This results in lower evaluation rates compared to unconstrained algorithms (Kuroiwa and Fukunaga 2020; Shimoda and Fukunaga 2023).

Separate Generation and Evaluation (SGE) (Shimoda and Fukunaga 2024b) is an implementation technique for parallel search which parallelizes state evaluations in addition to expansions. This allows better utilization of threads which would otherwise be idle (waiting for a state which satisfies expansion constraints). The main idea is to decompose the expansion of state s into separate units of work which can be parallelized: (1) successor generation, which generates the $succ(s)$, the successors of s, and (2) successor evaluation, which evaluates $succ(s)$.

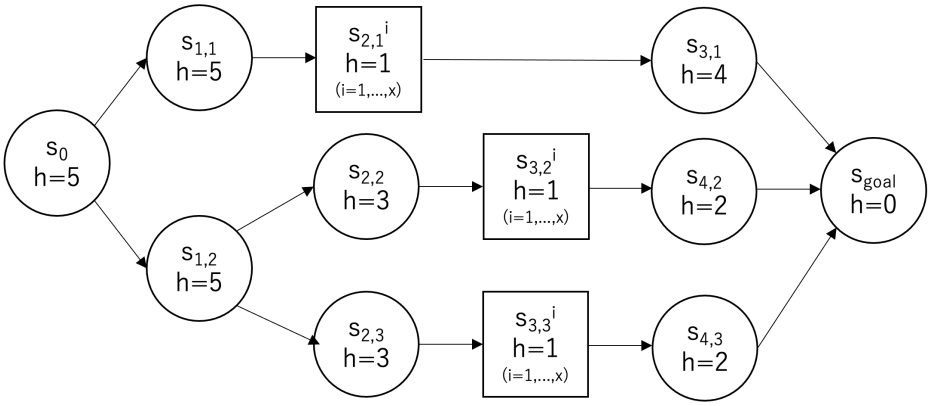
In SGE, after a thread selects a state for expansion from the shared $Open$, it generates $succ(s)$, and inserts $succ(s)$ into a shared $Unevaluated$ queue. The evaluation of states in $Unevaluated$ is done in parallel, taking precedence over selection of states for expansion (a thread will select a state for expansion from $Open$ only if $Unevaluated$ is currently empty). Evaluated states are *not* immediately inserted into $Open$. Instead, SGE inserts all members of $succ(s)$ of s simultaneously into $Open$, after they have all been evaluated This is so that the parallel search is able to prioritize $succ(s)$ similarly to GBFS (otherwise, $succ(s)$ can be expanded in a completely different order than by GBFS).

3 Pathological Behavior of PUHF

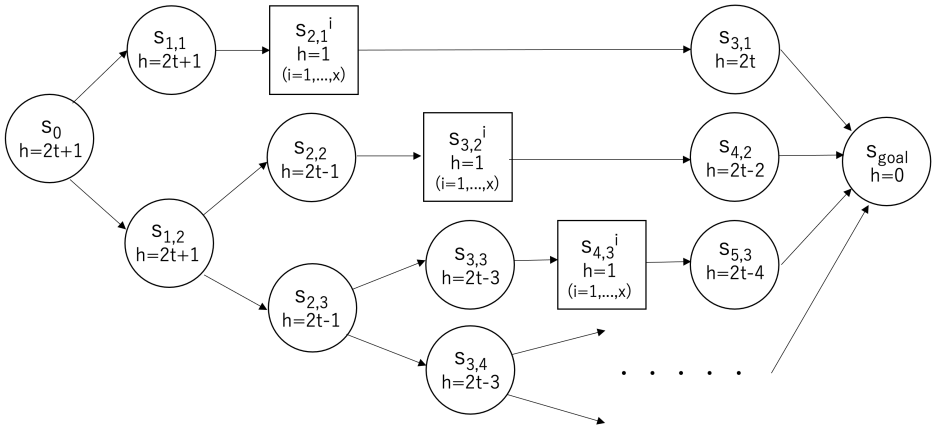
In this section, "PUHF" refers to PUHF (Kuroiwa and Fukunaga 2020) and PUHF2-4 (Shimoda and Fukunaga 2023) – although these PUHF variants use slightly different state expansion constraints, the arguments and conclusions below are unaffected by these differences.

While PUHF is guaranteed to only expand states in the BTS, previous work did not guarantee a bound on the number of states expanded relative to sequential GBFS. We show that in fact PUHF is pathological relative to sequential GBFS. The example we use in the proof motivates OBAT, the new bounded approach we propose in Section 4.

Consider the search space in Figure 1a. GBFS, under any tie-breaking policy must expand either $s_0 \rightarrow s_{1,1} \rightarrow \mathbf{s^i_{2,1}} \rightarrow s_{3,1} \rightarrow s_{goal}$, $s_0 \rightarrow s_{1,2} \rightarrow s_{2,2} \rightarrow \mathbf{s^i_{3,2}} \rightarrow s_{4,2} \rightarrow s_{goal}$, or



(a) PUHF Pathological Search Behavior Example 1



(b) PUHF Pathological Search Behavior Example 2

Figure 1: Pathological Search Behavior in PUHF

$s_0 \rightarrow s_{1,2} \rightarrow s_{2,3} \rightarrow \mathbf{s^i_{3,3}} \rightarrow s_{4,3} \rightarrow s_{goal}$ where $\mathbf{s^i}$ denotes a line of x states.

However, PUHF with $k \geq 2$ (k is the number of threads) may (depending on tie-breaking) expand states in the order: $s_0 \rightarrow (s_{1,1} \& s_{1,2}) \rightarrow \mathbf{s^i_{2,1}} \rightarrow (s_{2,2} \& s_{2,3}) \rightarrow (\mathbf{s^i_{3,2}} \cup \mathbf{s^i_{3,3}}) \rightarrow (s_{4,2} \& s_{4,3}) \rightarrow s_{goal}$ where $a \& b$ denotes that states a and b are simultaneously expanded. All of these states are in the BTS, and when they are expanded in this order, they satisfy the expansion criteria used by all previous variants of PUHF.

PUHF expands $3x$ states (denoted by the boxes $s_{2,1}$, $s_{3,2}$, $s_{3,3}$), as well as the 6 other non-goal states. For large x, PUHF with $k \geq 2$ may expand 3 times the number of states expanded by GBFS with the worst-case tie-breaking policy.

By recursively nesting the structure of the search space in Figure 1a, we can construct the search space of Figure 1b. On this search space, following the reasoning above for Figure 1a, GBFS with the worst-case tie-breaking policy will expand $t + x + c$ states (c is a constant). In contrast, PUHF may expand more than $x \cdot t$, where t can be made arbitrarily large. Thus, PUHF with $k \geq 2$ and any tie-breaking strategy is pathological relative to GBFS. PUHF is pathological relative to GBFS even if the heuristic is consistent. For example, consider an instance of the search space with the structure shown in Figure 1b, where all edge costs are 1. If we replace all h-values by h', such that $h' = h/c$, where c is a sufficiently large constant such that $h/c < 1$, then h' is consistent, but PUHF is pathological on this search space.

Why PUHF is pathological relative to GBFS even though it is BTS-constrained is that unlike GBFS, parallel algorithms such as PUHF can explore multiple benches simultaneously.

In Figure 1a, $s_{1,1}$ and $s_{1,2}$ are both progress states. The benches entered through $s_{1,1}$ and $s_{1,2}$ are both part of the BTS, and are candidates for exploration by GBFS. GBFS will expand either $s_{1,1}$ or $s_{1,2}$ and explore one of the corresponding benches, and cannot explore the other bench until exiting the first bench. On the other hand, PUHF can expand both $s_{1,1}$ and $s_{1,2}$, causing both benches to be explored. As a result, although PUHF only expands states in the BTS, it can explore sets of benches which are not explored by GBFS with any one specific tie-breaking strategy.

4 One Bench At a Time (OBAT)

We now propose One Bench At a Time (OBAT), a BTS-constrained parallel BFS which behaves similarly to GBFS

in that it is constrained to explore one bench at a time. The main idea is to enforce this constraint by carefully controlling the expansion of potential progress states so that if multiple potential progress states are expanded simultaneously, only the successors of one of these potential progress states are allowed to be inserted into $Open$. For example, if s_1 and s_2 are expanded simultaneously, and they are both potential progress states, then only the successors of s_1 are inserted into $Open$, because if both $succ(s_1)$ and $succ(s_2)$ are inserted into $Open$, we may start to explore multiple benches simultaneously.

OBAT classifies states into two categories, a-states and b-states. A state s is an a-state if it does not have any successor with a lower h-value than $h(s)$. An a-state cannot be a progress state (because if $s' \in succ(s)$ and $h(s) \leq h(s')$, then $hwm(s) \leq hwm(s')$, so if no successor of s has a lower h-value than s, then s cannot satisfy Definition 4). A state s is a b-state if it has a successor with a lower h-value than $h(s)$. A b-state is potentially a progress state. It is possible to determine whether a state is an a-state or b-state immediately after its successors are generated and their h-values are evaluated.

If s is an a-state, OBAT proceeds normally as with standard GBFS, inserting its successors in $Open$. However, if s is a b-state, it is a potential progress state, so instead of inserting the successors of s into $Open$, it inserts s into $Deferred$, a priority queue (priority function is h-value). A state s in $Deferred$ is "on hold" – its successors are not inserted in $Open$ until s is removed from $Deferred$. Thus, if multiple b-states are simultaneously expanded from $Open$, $Deferred$ prevents their successors from being simultaneously inserted into $Open$.

The sequence in which a state moves among the data structures in OBAT (i.e., its "life cycle") depends on whether it is an a-state or b-state. a-states are: (a1) generated and evaluated, (a2) inserted in $Open$ and $Closed$, (a3) removed from $Open$. b-states are: (b1) generated and evaluated, (b2) inserted in $Open$ and $Closed$, (b3) removed from $Open$, (b4) inserted in $Deferred$, (b5) removed from $Deferred$.

Definition 8. A state is *committed* after it leaves $Open$ or $Deferred$ for the last time. Specifically, a-states are committed after (a3), and b-states are committed after (b5).

Definition 9. A state s is *completely expanded* if it is committed and $succ(s)$ have all been inserted in $Open$.

Algorithm 2 shows the pseudocode for OBAT. $top(Q)$ returns a highest priority (minimal h-value, ties broken according to a tie-breaking policy) state from queue Q. By convention, $h(top(Q)) = \infty$ if Q is empty. For all threads in parallel, OBAT proceeds as follows.

When $h(top(Deferred)) \leq h(top(Open))$ (line 10) and $h(top(Deferred))$ is less than or equal to the minimal h-value among all states currently being expanded by other threads (line 11), we pop $top(Deferred)$ and insert its successors into $Open$, completing its expansion.

If $h(top(Open)) < h(top(Deferred))$ (line 20) and $h(top(Open))$ is less than or equal to the smallest h-value among states currently being expanded by other threads (line 21), we set s_i to $top(Open)$, pop $top(Open)$ (line 22), and generate $succ(s_i)$.

If s_i is an a-state (line 26), $succ(s_i)$ are inserted into $Open$ (so s_i is completely expanded). If s_i is a b-state (line 33), s_i is inserted into $Deferred$, and $succ(s_i)$ are *not* inserted anywhere, but they remain linked to s_i.

Lines 11 and 21 ensure that OBAT will not expand states with a higher h-value than any state which is currently expanded by another thread. This is the same expansion constraint as PUHF2, so it follows directly from (Shimoda and Fukunaga 2023, Theorem 4) that:

Proposition 1. OBAT is BTS-constrained.

Example of OBAT Search Behavior To illustrate the behavior of OBAT, consider an example run of OBAT with 2 threads on the search space in Figure 1a. First, one thread expands the initial state s_0. Since s_0 does not have any successors with a smaller h-value, $s_{1,1}$ and $s_{1,2}$ are both inserted in $Open$. Next, threads 1 and 2 remove $s_{1,1}$ and $s_{1,2}$ from $Open$, respectively and expand them. Since $s_{1,1}$ and $s_{1,2}$ both have successors with a smaller h-value than themselves, $s_{1,1}$ and $s_{1,2}$ are both potential progress states, so they are both inserted into $Deferred$. Next, a state is removed from $Deferred$. Assume $s_{1,1}$ is removed (the specific state depends on the tie-breaking policy). $s_{1,1}$ is completely expanded ($succ(s_{1,1})$ are inserted into $Open$). Then, both threads will search the line of states $s^1_{2,1}, .., s^x_{2,1}$, and then $s_{3,1}$, reaching the goal. Thus, whenever OBAT selects for expansion a potential progress state ($s_{1,1}$ and $s_{1,2}$ in this example), OBAT defers them, and will only completely expand one of them, thereby preventing simultaneous exploration of multiple benches.

5 Analysis of State Expansions by OBAT

We now compute a bound on the number of states expanded by OBAT. The number of states expanded by OBAT is the sum of (1) the number of completely expanded states and (2) the number of states remaining in $Deferred$ when the algorithm terminates.

Theorem 2. The number of states completely expanded by OBAT is less than or equal to the number of states expanded by GBFS with some tie-breaking policy.

Proof. Let s_1,s_2,s_3... be a serialized ordering of the states which are completely expanded by OBAT, in the order in which they are committed. We show that there exists a tie-breaking policy for sequential GBFS which expands states in this same order. The proof is by induction. For both sequential GBFS and OBAT, the first (completely) expanded state s_1 corresponds to the initial state s_{init}. Next, assume that for the ordering $s_1, ..., s_i$, there exists a tie-breaking policy for GBFS which expands states in that order. We show that there exists a tie-breaking policy for sequential GBFS which will expand s_{i+1} after s_i.

When sequential GBFS removes s_{i+1} from $Open$, the set of states in $Open$ are $\bigcup_{k=1}^{k=i} succ(s_k) \setminus \bigcup_{k=1}^{k=i} s_k$ (the set of all generated states minus the set of all expanded states). Now consider when OBAT commits s_{i+1}. Let A and B denote

Algorithm 2: One Bench At a Time (OBAT)

```
1:  Open ← {s_init}, Closed ← {s_init}; ∀i, s_i ← NULL
2:  for i ← 0, ..., k − 1 in parallel do
3:      loop
4:          while s_i = NULL do
5:              lock(Open), lock(Deferred)
6:              if Open = ∅ and Deferred = ∅ then
7:                  if ∀j, s_j = NULL then
8:                      unlock(Open), unlock(Deferred); return NULL
9:              else
10:                 if h(top(Deferred)) ≤ h(top(Open)) then
11:                     if h(top(Deferred)) ≤ min_{0≤j<k} h(s_j) then
12:                         s ← top(Deferred)
13:                         Deferred ← Deferred\{s}
14:                         lock(Closed)
15:                         for s' ∈ succ(s) do
16:                             if s' ∉ Closed then
17:                                 Closed ← Closed ∪ {s'}
18:                                 Open ← Open ∪ {s'}
19:                         unlock(Closed)                          ▷ s is completely expanded
20:                 else
21:                     if h(top(Open)) ≤ min_{0≤j<k} h(s_j) then
22:                         s_i ← top(Open); Open ← Open\{s_i}
23:             unlock(Open), unolock(Deferred)
24:         if s_i ∈ S_goal then
25:             return Path(s_i)
26:         if h(s_i) ≤ min(h(succ(s_i))) then                      ▷ s_i is an a-state
27:             lock(Open), lock(Closed)
28:             for s'_i ∈ succ(s_i) do
29:                 if s'_i ∉ Closed then
30:                     Closed ← Closed ∪ {s'_i}
31:                     Open ← Open ∪ {s'_i}
32:             unlock(Open), unlock(Closed)                        ▷ s_i is completely expanded
33:         else                                                    ▷ s_i is a b-state
34:             lock(Deferred)
35:             Deferred ← Deferred ∪ {s_i}
36:             unlock(Deferred)
37:         s_i ← NULL
```

the set of a-states and b-states currently being expanded by other threads. Let A' denote the set of successors of all states in A. $\bigcup_{k=1}^{k=i} succ(s_k) \setminus \bigcup_{k=1}^{k=i} s_k = (Open \cup Deferred \cup B \cup (A' \setminus \bigcup_{k=1}^{k=i} s_k))$. s_{i+1} has the smallest h-value among all states in $Deferred \cup Open$.

$A \cup B$ cannot contain any states taken from $Deferred$, because $Open$ and $Deferred$ are locked while a thread removes a state from $Deferred$ and inserts its successors into $Open$, so this cannot occur simultaneously with the complete expansion of s_{i+1}, which requires access to $Open$ and $Deferred$. Thus, all states in $A \cup B$ were taken from $Open$. All states in B will later be inserted in $Deferred$, so are not in $s_1, ...s_i$. All states in A will be included in $s_1, ..., s_i$. By definition, a-states do not have any successors with a smaller h-value than itself.

Due to lines 11 and 21, $h(s_{i+1}) \leq h(c) \forall c \in (A \cup B)$. Therefore, $h(s_{i+1}) \leq min_{a \in A} h(a) \leq min_{u \in A'} h(u)$. Thus, $h(s_{i+1})$ has the smallest h-value among $(Open \cup Deferred \cup B \cup (A' \setminus \bigcup_{k=1}^{k=i} s_k))$.

This implies, s_{i+1} has the smallest h-value in $Open$ after GBFS expands s_i, so there exists a tie-breaking policy for sequential GBFS which will select s_{i+1} for expansion immediately after s_i. Thus, there exists a tie-breaking policy for GBFS which expands states in the same order as they are committed in OBAT. Therefore, the number of states completely expanded by OBAT is less than or equal to the number of states expanded by GBFS with the worst-case tie-breaking policy. □

To bound the number of states in $Deferred$ when OBAT terminates, we bound the number of different h-values for which there is a state in $Deferred$ (Lemma 1) and the number of states per h-value (Lemma 2).

Lemma 1. When OBAT terminates after finding a solution path p, the number of h-values for which at least one state is in *Deferred* is at most $|p|$.

Proof. Let $p = s_1, s_2, ..., s_n$ be the sequence of all states on the solution path (in the order they appear in the solution path), such that all states after s_i on the solution path have a smaller h-value than s_i. From the definition of p, all states in the solution path between s_i and s_{i+1} have an h-value less than or equal to $h(s_{i+1})$. Thus, after the successors of s_i are inserted in *Open*, there is a state with h-value less than or equal to $h(s_{i+1})$ in *Open*, *Deferred*, or a state currently being expanded by some thread. OBAT will not select a state from *Open* with a h-value larger than a state currently being expanded by some thread (lines 11 and 21), so between the time after the successors of s_i are inserted in *Open* and s_{i+1} is completely expanded, no states with h-values larger than $h(s_{i+1})$ will be removed from *Open*.

States with h-value less than $h(s_{i+1})$ will have higher priority for expansion than s_{i+1}, so no such state can be in *Deferred* when s_{i+1} is expanded. Thus, in the time between when s_i is completely expanded, and when s_{i+1} is removed from *Deferred* and committed, the only states which can be removed from *Open* and inserted into *Deferred* (and can remain in *Deferred* after OBAT terminates) have an h-value of $h(s_{i+1})$. Similarly, for the base case (s_1), the only states which can be removed from *Open* before s_1 and might remain in *Deferred* after OBAT terminates have a h-value of $h(s_1)$. Therefore, the only possible h-values of states in *Deferred* when OBAT terminates are $h(s_1), h(s_2), ..., h(s_n)$, which is $\leq n$ unique values. □

Lemma 2. At all times, for every h-value, *Deferred* contains at most k states, where k is the number of threads.

Proof. Assume that *Deferred* contains $k+1$ states which all have the same h-value. By the pigeonhole principle, there must be 2 such states s_1 and s_2, which were inserted into *Deferred* by the same thread T, where s_1 was inserted before s_2. States in *Deferred* come from *Open*, i.e., a state is inserted in *Deferred* only after it is removed from *Open*. Therefore, for s_2 to be inserted into *Deferred*, it must first be removed from *Open*, and by assumption, at that time, s_1 is already in *Deferred*. However, if $h(top(Open)) = h(top(Deferred))$, then OBAT will remove a state from *Deferred* (Algorithm 2, line 12), so $s_2 \in$ *Open* will not be removed before $s_1 \in$ *Deferred*, a contradiction. □

Lemma 1 and Lemma 2 directly give us a bound on the number of states in *Deferred* when OBAT terminates.

Theorem 3. When OBAT terminates after finding a solution path p, the number of states which remain in *Deferred* is less than or equal to $k|p|$, where k is the number of threads.

Finally, from Theorem 2 and Theorem 3, we have the following bound on the number of states expanded:

Theorem 4. The number of states expanded by OBAT is at most $N_{gbfs} + k|p|$, where N_{gbfs} is the number of states expanded and p is the solution path found by GBFS with the tie-break policy which maximizes $N_{gbfs} + k|p|$.

As an aside, since $|p| \leq N_{gbfs}$, OBAT is t-bounded relative to GBFS with the tie-break policy which maximizes $N_{gbfs} + k|p|$, where $t = k+1$. However, N_{gbfs} is usually much larger than $|p|$, so the bound in Theorem 4 is more meaningful than the t-boundedness.

6 OBAT$_S$: OBAT with SGE

Like PUHF and its variants, OBAT is a constrained parallel GBFS, and threads can be forced to be idle while they wait until a state which is guaranteed to satisfy the expansion constraint becomes available, resulting in worse performance than unconstrained algorithms such as KPGBFS which never leave threads idle. As with PUHF, this issue can be alleviated using Separate Generation and Evaluation (Shimoda and Fukunaga 2024b), an implementation technique which decouples successor generation and evaluation to reduce idle waiting.

Applying SGE to OBAT is straightforward. Algorithm S.1 in the Supplement (Shimoda and Fukunaga 2024a) shows OBAT$_S$, which is OBAT with SGE. As SGE does not change the set of states which are possibly expanded, it can be proven straightforwardly that the upper bound on the number of states expanded by OBAT (Theorem 4) also holds in OBAT$_S$.

7 Experimental Evaluation

We evaluated the performance of OBAT and OBAT$_S$ with 4, 8, 16 threads. As baselines, we also evaluate KPGBFS, KPGBFS$_S$ (KPGBFS with SGE), PUHF3, PUHF3$_S$ (PUHF3 with SGE) , and single-thread GBFS.

Experimental Settings We compared 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), an improved benchmark suite based on the IPC classical planning benchmarks. However, for domains where (1) all methods solved all instances (i.e., the instances were too easy for the purpose of comparing the performance of KPGBFS vs. OBAT), 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, the domains where criteria (1) and (2) above applied were gripper and miconic. See Supplement Section S.2 (Shimoda and Fukunaga 2024a) for additional details, including instance generator parameters.

All algorithms use the FF heuristic (Hoffmann and Nebel 2001). Each run had a time limit of 5 min., 3GB RAM/thread (e.g., 8 threads: 24GB total) limit on an Intel Xeon CPU E5-2670 v3@2.30GHz processor. All tie-breaking is FIFO. The code is available at https://github.com/TakuShimoda/AAAI25.

Results Table 1a shows the total coverage (number of instances solved out of 1260) for 4/8/16 threads. Table 1b shows per-domain coverage for 16 threads. Figure 2 compares the number of states expanded by the algorithms. Figure 3 compares the state evaluation rates of the algorithms.

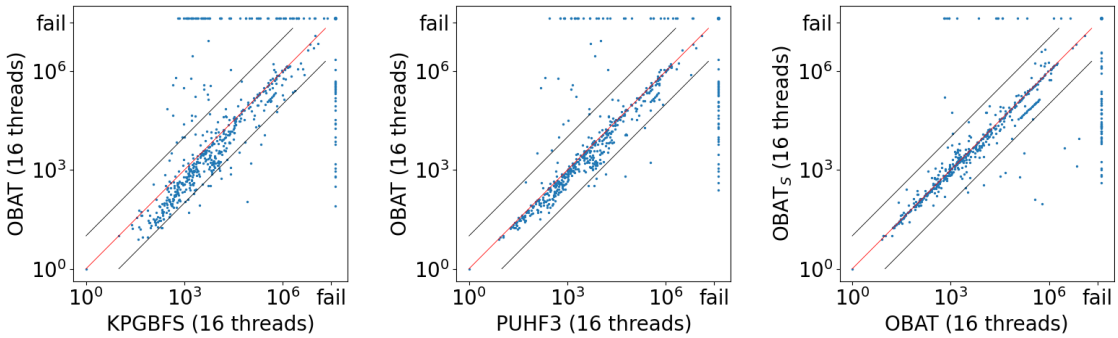


Figure 2: Number of states expanded, 16 threads. Diagonal lines are $y = 0.1x$, $y = x$, and $y = 10x$

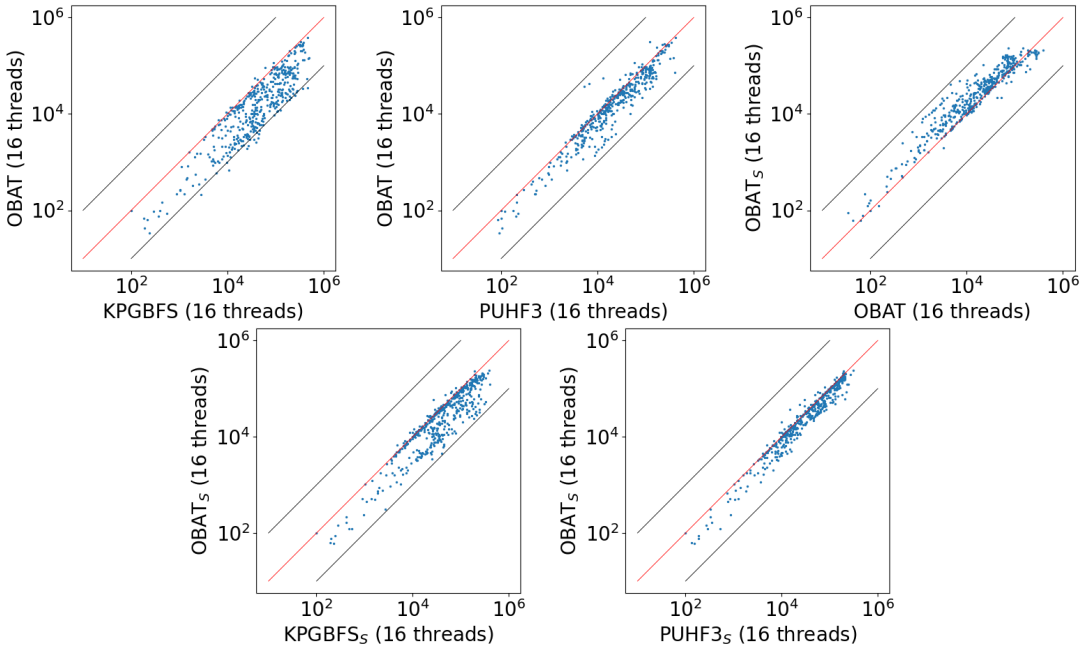


Figure 3: State evaluation rates (states/second), 16 threads. Diagonal lines are $y = 0.1x$, $y = x$, and $y = 10x$

Additional figures comparing number of state expansions, state evaluation rates, and search time including data for 4 and 8 threads, are in the Supplement, Figures 1-6.

Compared to KPGBFS, OBAT expands significantly fewer states (Figure 2, left). However, OBAT has significantly lower state evaluation rates than KPBGFS (Figure 3 top left). This results in significantly lower total coverage than KPGBFS (Table 1a), although the difference is domain-dependent, e.g., OBAT significantly outperforms KPGBFS on openstacks. Compared to PUHF3, OBAT also expands fewer states (Figure 2, middle) has a lower evaluation rate (Figure 3 top right), and has comparable total coverage. Thus, the expansion constraint appears to enable OBAT to search good regions of the search space, at the cost of low evaluation rates.

$OBAT_S$ has a significantly higher evaluation rate than OBAT (Figure 3, bottom left) and a comparable number of state expansions (Figure 2, right), showing that SGE significantly alleviates the drawback of OBAT without sacrificing search efficiency. As a result, $OBAT_S$ achieves significantly higher coverage than OBAT (Table 1a).

Overall, $OBAT_S$ achieves coverage competitive with all other methods for 4/8/16 threads, showing that the guaranteed bound on the number of expansions does not require sacrifing practical performance. As shown in Table 1b, none of the algorithms clearly dominate the others, and comparative performance is domain-dependent. Despite the state evaluation rate boost due to SGE, $OBAT_S$ still has a significantly lower evaluation rate than $KPGBFS_S$ and $PUHF3_S$ (Figure 3, bottom middle and right) – further decreasing the evaluation rate gap (improving thread utilization) while maintaining search efficiency is a direction for future work.

8 Conclusion

We proposed OBAT, a parallel GBFS algorithm which, to our knowledge, is the first parallel GBFS with a shared *Open* list that guarantees a bound on the number of states

#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**

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

	GBFS	KPGBFS	KPGBFS$_S$	PUHF3	PUHF3$_S$	OBAT	OBAT$_S$
agricola	26	30	30	30	30	30	30
airport	18	17	18	17	**19**	15	18
barman	2	4	4	4	4	4	**5**
blocksworld	6	7	7	7	**8**	6	7
childsnack	4	**5**	3	**5**	4	4	4
data-network	4	**7**	6	6	4	5	5
depots	4	**5**	**5**	4	4	4	4
driverlog	4	7	8	8	8	7	**9**
elevators	11	18	19	**20**	**20**	**20**	19
floortile	2	2	2	2	2	2	2
freecell	20	24	23	21	22	22	**25**
ged	7	9	9	7	8	**10**	8
grid	5	5	**6**	**6**	**6**	**6**	**6**
gripper (replaced)	2	7	**21**	7	20	19	20
hiking	4	**14**	9	11	8	10	9
logistics	4	**6**	**6**	5	5	**6**	**6**
miconic (replaced)	10	9	15	9	15	10	**21**
mprime	4	**6**	**6**	**6**	**6**	5	5
nomystery	6	**14**	11	11	11	3	6
openstacks	7	12	11	11	11	15	**22**
organic-synthesis-split	9	**17**	**17**	16	16	13	16
parcprinter	30	30	30	30	30	30	30
parking	6	9	**10**	9	9	9	**10**
pathways	11	**15**	12	10	12	11	12
pegsol	30	30	30	30	30	30	30
pipesworld-notankage	8	**11**	9	9	9	9	**11**
pipesworld-tankage	9	**12**	**12**	11	11	**12**	**12**
rovers	26	**27**	**27**	26	26	26	26
satellite	7	7	7	7	7	7	7
scanalyzer	7	11	11	11	11	11	11
snake	7	**10**	**10**	9	9	9	9
sokoban	16	**23**	22	21	21	19	19
storage	3	2	**4**	**4**	**4**	3	**4**
termes	12	**18**	**18**	16	**18**	15	16
tetris	8	**13**	12	**13**	12	**13**	**13**
thoughtful	14	**21**	18	19	16	16	14
tidybot	12	**14**	13	12	11	**14**	13
tpp	8	10	11	9	10	11	**12**
transport	5	**8**	7	**8**	7	7	7
visitall	14	**20**	19	13	14	15	16
woodworking	2	2	2	2	2	2	2
zenotravel	7	11	**12**	**12**	10	11	11
Sum(1260)	401	529	**532**	494	510	496	**532**

(b) Coverage (number of problems solved) per domain on Autoscale-21.11 IPC-based planning benchmark set (1260 instances total; gripper, and miconic are replaced with harder instances. See S.2) for k=16 threads (except k=1 for GBFS).

Table 1: Coverage results

expanded relative to sequential GBFS. The worst-case gap in the number of state expansions between OBAT and sequential GBFS is the gap between the best-case sequential GBFS tie-breaking policy and the worst-case sequential GBFS tie-breaking policy.

These results establish a close relationship between parallel greedy best-first search and theoretical work in tie-breaking policies and opens interesting avenues for future work – for example, any upper bound on the gap between best vs. worst-case tie-breaking will also imply an upper bound on the gap between sequential and parallel greedy best-first search.

We showed experimentally that while OBAT achieves search efficiency competitive with previous shared-$Open$ parallel GBFS algorithms, the state evaluation rate was significantly lower than previous algorithms. Finally, we showed that applying SGE to OBAT significantly increases the state evaluation rate of OBAT, resulting in overall performance which is comparable to previous methods, at least up to 16 threads. Investigating and improving scalability to larger number of threads is a direction for future work.

Acknowledgments

This research was supported by JSPS KAKENHI Grant 24K15083.

References

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.

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.

Heusner, M.; Keller, T.; and Helmert, M. 2018. Best-Case and Worst-Case Behavior of Greedy Best-First Search. In *Proc. IJCAI*, 1463–1470.

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

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. 2023. Improved Exploration of the Bench Transition System in Parallel Greedy Best First Search. In *Proc. SOCS*, 74–82.

Shimoda, T.; and Fukunaga, A. 2024a. Parallel Greedy Best-First Search with a Bound on the Number of Expansions Relative to Sequential Search. Extended version of this paper (includes Supplement), arXiv:2412.12221.

Shimoda, T.; and Fukunaga, A. 2024b. Separate Generation and Evaluation for Parallel Greedy Best-First Search. In *Proceedings of ICAPS 2024 Workshop on Heuristics and Search for Domain-Independent Planning (HSDIP)*.

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.

Wilt, C. M.; and Ruml, W. 2014. Speedy Versus Greedy Search. In *Proc. SOCS*, 184–192.