Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Adaptive Lower Bound Evaluation for the Permutation Flowshop Scheduling Problem

Alisa Vorokhta

alisa.vorokhta@uni.lu

University of Luxembourg

Jan Gmys

University of Lille

Gwen Maudet

University of Luxembourg

Mohand Mezmaz

University of Luxembourg

Grégoire Danoy

University of Luxembourg

Research Article

Keywords: Permutation Flowshop Scheduling Problem, Branch-and-bound, Lower Bounds, Machine-pair Selection, Two-machine Relaxation

Posted Date: August 28th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-10154547/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

Adaptive Lower Bound Evaluation for the Permutation Flowshop Scheduling Problem

Alisa Vorokhta[1*], Jan Gmys[2†], Gwen Maudet[1†], Mohand Mezmaz[1†], Grégoire Danoy[3,1†]

[1*]SnT, University of Luxembourg, Avenue de l'Université, Esch-sur-Alzette, L-4365, Luxembourg.
[2]Mésocentre de Lille, University of Lille, Rue Paul Duez, Lille, 59800, France.
[3]FSTM/DCS, University of Luxembourg, Avenue de l'Université, Esch-sur-Alzette, L-4365, Luxembourg.

*Corresponding author(s). E-mail(s): alisa.vorokhta@uni.lu;
Contributing authors: jan.gmys@univ-lille.fr; gwen.maudet@uni.lu;
mohand.mezmaz@uni.lu; gregoire.danoy@uni.lu;
†These authors contributed equally to this work.

Abstract

Background: The two-machine bound $\boldsymbol{LB_2}$ in permutation flowshop scheduling problem (PFSP) branch-and-bound (B&B) is used to evaluate the quality of a partial solution and it is computed over multiple machine pairs. The evaluation can be stopped earlier if pruning is feasible. As a result, the selection of pairs and their arrangement are important. More pairs imply better bounds but slower computation, and fewer pairs reduce costs but may weaken pruning. The trade-off has a large effect on performance.

Methods: In this work, we provide the first systematic analysis of the selecting machine pairs problem for computing $\boldsymbol{LB_2}$. We extend the adaptive mechanism for machine-pair selection and propose limiting the candidate set to machine pairs that include the first or the last machine, reward sharing when multiple machine pairs yield the same bounds, and alternative scoring such as exponentially weighted moving averages (EWMA) ranking and one that combines accumulated reward with a pair's mean lower bound. In addition, we implement algorithms that dynamically modify the number of pairs assessed at each node. Finally, we evaluate their empirical impact.

Results: Experiments on Taillard benchmark instances demonstrate a trade-off between bound tightness and runtime. Restricted strategies significantly speed up the search compared to the full version of $\boldsymbol{LB_2}$. At the same time, adaptive approaches using all machine pairs remain a strong baseline, while methods dynamically adjusting the active set often achieve similar performance on harder instances. Random strategies tend to be less reliable.

Keywords: Permutation Flowshop Scheduling Problem; Branch-and-bound; Lower Bounds; Machine-pair Selection; Two-machine Relaxation

1 Introduction

The exact methods for scheduling problems remain important whenever an optimal solution with proof of optimality is required. This is particularly relevant for benchmarking heuristic approaches and for solving problem instances where optimality guarantees can be critical. In such settings, branch-and-bound (B&B) algorithms are among the standard exact approaches. They systematically explore the search space and prune subproblems whose bounds show that they cannot improve the incumbent solution (Morrison et al. (2016); Brucker (2007); Ruiz and Maroto (2005)).

For minimization problems, a node can be pruned when its lower bound (LB) is greater than or equal to the current upper bound (UB), i.e. the objective value of the best feasible solution found so far. As a result, bound quality and computational cost are tightly coupled. Simple bounds are inexpensive to compute, but may provide limited pruning, while stronger, tighter LBs can significantly reduce the search tree at the expense of greater computational effort per node (Morrison et al. (2016)).

This trade-off is clearly visible in the permutation flowshop scheduling problem (PFSP), where each job must visit all machines in the same order. In this case, two-machine relaxations provide effective lower bounds, commonly referred to as LB_2 (Lageweg et al. (1978); Ladhari and Haouari (2005)). The classical LB_2 bound is obtained by solving a relaxation for a pair of machines and taking the maximum value over a set of machine pairs (Lageweg et al. (1978)). However, evaluating all $\binom{m}{2}$ machine pairs can become costly as the number of machines grows, especially compared to a simple LB computation based on a single selected pair.

This is not just a theoretical concern: in our analysis, the evaluation of machine pairs accounts for approximately 89%–98% of the per-node LB_2 time, while the shared preprocessing contributes only about 2%–11% (see Section 5.4).

In addition, different machine pairs may produce bounds of very different quality. Consequently, the order in which pairs are evaluated can influence the number of pairs computed before pruning occurs, and therefore the total runtime.

1.1 Motivation

These observations raise a natural question: *how should a B&B solver choose and order machine pairs when computing LB_2, especially when early exit is possible?* Here, *early exit* refers to stopping the evaluation of machine pairs as soon as the current lower bound reaches or exceeds the incumbent upper bound, since the node can then be pruned (Land and Doig (1960)).

The effectiveness of a selection policy depends not only on the average bound quality but also on how quickly it tends to find a pair that is strong enough to trigger pruning, how stable the best-performing pairs are across the search tree, and how much overhead is introduced by the selection mechanism itself.

1.2 Contributions

In this work, we study the problem of selecting machine pairs for computing LB_2 and its impact on both bound quality and computational efficiency. In our earlier work (Gmys et al. (2020)), we introduced a lightweight adaptive evaluation mechanism for machine-pair selection as a preliminary idea, without an in-depth analysis. This paper systematically develops this direction. More specifically, we compare classical static strategies with an online adaptive mechanism, design and evaluate different practical variants, and quantify their impact in a extensive empirical evaluation. The variants considered include tie-aware reward sharing when multiple pairs yield identical bounds, restricting the candidate set to extreme machine pairs, alternative scoring mechanisms that replace standard reward-based ranking, and mechanisms for dynamically adjusting the number of top-ranked pairs evaluated.

Our contributions are mainly empirical and practical:

- We study machine-pair selection as a trade-off between bound strength and computational cost in early-exit LB_2 evaluation.
- We extend our lightweight online mechanism (Gmys et al. (2020)) with tie-aware/restricted variants, alternative scoring, and a dynamic choice of how many top-ranked pairs to evaluate.
- We provide a sensitivity analysis of the learning parameters and suggest default settings

that perform well across the tested Taillard benchmark sizes.

Overall, our results highlight that evaluating more machine pairs does not necessarily lead to better overall performance. In many cases, limiting the number of pairs evaluated while maintaining an effective ordering can significantly reduce the overall runtime of B&B.

1.3 Organization of the paper

The remainder of the paper is organized as follows. Section 2 briefly reviews PFSP, B&B, and the classical bounds LB_1 (a one-machine LB) and LB_2 (Lageweg et al. (1978)). Section 3 discusses related work. Section 4 describes the compared machine-pair selection strategies. Section 5 presents the experimental setup and results. Section 6 concludes the paper, and supplementary tables are provided in Appendix A.

2 Background

We first introduce the PFSP and the B&B framework, and then summarize the lower bounds LB_1 and LB_2 that form the basis of our study. We focus on the components needed to understand the bounding procedure and the overall search process.

2.1 PFSP

The PFSP is a classic NP-hard optimization problem where a set of n jobs must be processed on m machines. Each job follows the same fixed order of machines $M_1, \ldots, M_m$, and at any given time each machine can handle at most one job. Once an operation starts, it cannot be interrupted. The main goal is typically to minimize the makespan, i.e., the time when the last job finishes on the final machine (Brucker (2007)).

Figure 1 illustrates a small PFSP instance with four jobs and four machines using a Gantt chart representation. The schedule corresponds to a permutation of the jobs that is respected on every machine.

PFSP is one of the standard benchmark problems in scheduling and is widely used to evaluate exact and heuristic methods (Brucker (2007); Taillard (1993); Ruiz and Maroto (2005)). Even for small instances, minimizing the makespan is computationally challenging. Although the permutation requirement simplifies the overall structure of the problem, it introduces strong dependencies in the ordering of jobs.

This leads to an explosion in the number of possible schedules, which increases factorially with the number of jobs ($n!$). As a result, exact methods commonly rely on B&B algorithms, which use lower bounds (Morrison et al. (2016)) to eliminate parts of the search tree that cannot have an optimal solution. The effectiveness of these approaches strongly depends on the quality of the lower bounds, as tighter bounds allow for more aggressive pruning before exploration.

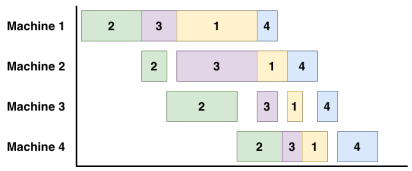


Fig. 1 Example Gantt chart of an optimal PFSP schedule with 4 jobs and 4 machines.

2.2 B&B

B&B is a standard exact framework for solving NP-hard combinatorial problems, including PFSP (Brucker (2007)). The classical scheme goes back to the original B&B method of Land and Doig (1960), which combines systematic branching with pruning based on bounding.

In the following, we focus on its components in the context of permutation-based scheduling minimization problems. A typical B&B algorithm for such types of problem repeatedly performs the following steps:

- *Branching*: generate child nodes by fixing an additional decision in the partial permutation. For PFSP, a typical scheme fixes the job placed at the next position from the beginning (building a prefix), and some variants may also fix jobs from the end (building a suffix).
- *Bounding*: compute an LB by approximating (from below) the best objective value reachable from the node.
- *Selection*: choose the next open node to process (e.g., depth-first or best-first based on node depth or bound value).

- *Pruning*: discard nodes whose LB is not better than the incumbent UB.

Here, the LB provides a guaranteed underestimate of the best achievable objective value from a partial solution, while the UB corresponds to the cost of the best feasible solution found so far. The pruning rule follows directly from this relationship: a node can be discarded whenever its LB is not better than the current UB, which in PFSP is commonly written as LB $\geq$ UB (Morrison et al. (2016)). In such exact solvers, the computation of bounds often dominates the per-node runtime, which makes the design of effective and inexpensive lower bounds crucial.

2.3 Lower-bound computation

Many LBs of the PFSP are obtained by relaxing the m-machine problem into exactly solvable one- or two-machine subproblems. In this work, we focus on LB_1 and LB_2, and on how machine-pair selection affects both the cost and the effectiveness of LB_2.

We represent a B&B node by two fixed subsequences: a prefix σ_1 (jobs fixed from the beginning of the permutation) and a suffix σ_2 (jobs fixed from the end), with the remaining set of unscheduled jobs denoted by J. Following the classical PFSP B&B notation (Gmys et al. (2020); Lageweg et al. (1978)), we use two standard terms evaluated on each machine M_k:

- a *head* term $r(\sigma_1, k)$ that captures how much time is already "spent" on M_k due to the fixed prefix σ_1;
- a *tail* term $q(\sigma_2, k)$ that corresponds to the processing time after M_k under the fixed suffix σ_2.

Intuitively, $r(\sigma_1, k)$ represents the completion time of the prefix σ_1 on machine M_k, i.e., the accumulated processing up to M_k. In contrast, $q(\sigma_2, k)$ captures the remaining processing induced by the suffix σ_2 on machines M_k to M_m. Standard conventions are used when either σ_1 or σ_2 is empty.

2.3.1 LB_1

For each machine M_k we denote by $p(k) = \sum_{j \in J} p_{j,k}$ the total remaining processing time on M_k, where $p_{j,k}$ is the processing time of job j on machine M_k. The one-machine bound then takes the worst (largest) machine-wise sum:

$$LB_1(\sigma_1, \sigma_2) = \max_{1 \leq k \leq m} \big(r(\sigma_1, k) + p(k) + q(\sigma_2, k)\big). \tag{1}$$

Evaluating LB_1 for a single node requires computing $r(\sigma_1, k)$, $q(\sigma_2, k)$, and $p(k)$ for all $k \in \{1, \ldots, m\}$, which can be done in $O(mn)$ add/max operations for n jobs. Moreover, if the algorithm evaluates LB_1 for all direct children of a parent node, this can still be performed in overall $O(mn)$ time by computing $r(\sigma_1, \cdot)$, $q(\sigma_2, \cdot)$, and $p(\cdot)$ once for the parent (in $O(mn)$), and then updating the bound for each of the $O(n)$ children in $O(m)$ time.

Due to its low computational cost, LB_1 is often used as a first filter in branch-and-bound: nodes satisfying $LB_1 \geq UB$ can be pruned without evaluating more expensive relaxations (Morrison et al. (2016)).

LB_1 can also be interpreted as a degenerate case of the two-machine bound LB_2, where both machines in the pair are identical ($k = l$) and no intermediate machines are considered. In this sense, LB_2 generalizes LB_1 by explicitly modeling interactions between distinct machine pairs (Lageweg et al. (1978)).

2.3.2 LB_2

A stronger bound is obtained by selecting two bottleneck machines M_k and M_l ($1 \leq k < l \leq m$) and relaxing capacity constraints on all other machines (Lageweg et al. (1978)). Let $C^{k,l}_{\max}(J)$ denote the optimal makespan of the resulting two-machine relaxation for unscheduled jobs solved by a Johnson-type ordering with appropriate heads, tails, and lags (time delays equal to the aggregated processing time of each job on the intermediate machines between M_k and M_l) (Johnson (1954); Lageweg et al. (1978)). Then

$$LB_2(\sigma_1, \sigma_2) = \max_{1 \leq k < l \leq m} \big(r(\sigma_1, k) + C^{k,l}_{\max}(J) + q(\sigma_2, l)\big). \tag{2}$$

LB_2 processes machine pairs sequentially rather than all machine pairs at once. This allows for an early exit, since the evaluation can be stopped as soon as a single pair is sufficient to prove that the node cannot lead to an improved solution. Specifically, if for some pair $LB_2(\sigma_1, \sigma_2) \geq UB$, the node can be pruned and the evaluation stops according to the standard

B&B principle (Land and Doig (1960)). As a result, the actual cost of computing LB_2 depends not only on the number of machine pairs considered, but also on the order in which they are evaluated. Processing stronger pairs earlier increases the likelihood of triggering an early exit and can therefore significantly reduce the average evaluation time.

Although solving a two-machine relaxation with Johnson's rule is often reported as $O(n \log n)$ because of the sorting step according to Johnson (1954), in our implementation the Johnson order for each machine pair is precomputed and stored. Consequently, once the $O(mn)$ head and tail values have been computed, evaluating a machine pair at a node only requires processing the remaining jobs in the corresponding Johnson order. Since this order is precomputed and stored, the evaluation takes $O(n)$ time per machine pair. When bounds are computed for all $O(n)$ children of a parent node, the overall complexity of LB_2 becomes $O(|W|n^2)$ for $|W|$ evaluated machine pairs, resulting in a quadratic dependence on n. Under the same evaluation scheme, LB_1 remains linear in n.

3 Related work

The exact algorithms for the PFSP have long relied on B&B, where the efficiency of the search is largely determined by the quality and cost of the LBs (Brucker (2007); Morrison et al. (2016)). For a recent overview of B&B methods for PFSP, we refer to the work of Tomazella and Nagano (2020). Over time, research in this area has explored increasingly strong bounds along with practical mechanisms that make them affordable in large search trees (Lageweg et al. (1978); Gmys et al. (2020)). At the same time, implementation choices, such as evaluation order, pruning strategies, and parallel execution, have been shown to interact closely with the cost of bound computations as shown in the works by Mezmaz et al. (2013); Melab et al. (2012); Helbecque et al. (2023). The works most closely related to our study lie at the intersection of these directions.

3.1 LBs for PFSP B&B

An important consideration in exact PFSP algorithms is the balance between bound strength and computational cost. Surveys on B&B emphasize that the effectiveness of a bound is determined not only by how tight it is, but also by how often and how early it enables pruning during the search (Morrison et al. (2016); Tomazella and Nagano (2020)).

In PFSP B&B solvers, LB_1 (Brown and Lomnicki (1966)) is often used as the cheapest bound: it can serve either as a standalone LB under tight per-node budgets or as a quick preliminary filter before investing time in two-machine evaluations. Its main limitation is structural, since it is derived from single-machine relaxations. Therefore, it cannot capture sequencing effects arising from interactions between machines. PFSP solvers typically combine inexpensive filters with stronger bounds applied more selectively Ladhari and Haouari (2005). The one-machine bound LB_1 is commonly used as a fast preliminary test, while the two-machine bound LB_2 (Lageweg et al. (1978)) remains one of the strongest polynomial-time bounds for makespan minimization and is widely used in exact approaches. Moreover, the choice of bound often depends on the region of the search tree.

Benchmark-based evaluation also plays an important role in this literature. The Taillard instances which are described in the work of Taillard (1993) have become a standard reference set, and are widely used in PFSP studies to enable controlled comparisons across different problem sizes and solution methods (Ruiz and Maroto (2005)).

3.2 Machine-pair restrictions for LB_2

In exact PFSP algorithms, a limitation of LB_2 is that evaluating all machine pairs incurs quadratic computational cost in the number of machines. Let $W_0 = \{(k, l) : 1 \leq k < l \leq m\}$ denote the full set of candidate machine pairs. The complete evaluation computes $LB_2(W_0)$ considering every pair in W_0 and is therefore the strongest among the variants $LB_2(W)$, but it can solve up to $\binom{m}{2}$ two-machine subproblems at a single node (Lageweg et al. (1978)). Implementations rely on constant-factor optimizations, including precomputing at the root the Johnson ordering of jobs for every machine pair, as proposed by Johnson Johnson (1954), so that evaluating a fixed pair (k, l) at a

Table 1 Summary of common machine-pair restrictions and adaptive evaluation policies for LB_2.

Policy	Pairs	Main idea / effect
Full evaluation — *AP*	$\mathcal{O}(m^2)$	Strongest variant, but quadratic work in the number of machines m.
Fixed-machine (e.g., last machine) — *Last*	$\mathcal{O}(m)$	Evaluate only pairs that include a designated machine to reduce cost.
Consecutive pairs — *Adj*	$\mathcal{O}(m)$	Restrict to adjacent machine pairs to keep structured coverage with linear work.

node mainly reduces to scanning that order for the currently unscheduled jobs.

To further reduce the cost, many practical variants evaluate only a restricted subset of pairs, accepting a potential loss in bound strength in exchange for faster computation. Two classical linear-size restrictions are:

$$W_1 = \{(k, k+1) : k = 1, \ldots, m-1\} \quad \text{(adjacent pairs (Nabeshima (1967))),} \tag{3}$$

$$W_2 = \{(k, m) : k = 1, \ldots, m-1\} \quad \text{(fixed last machine (Lageweg et al. (1978))).} \tag{4}$$

The consecutive-pair restriction keeps only local interactions and may miss long-range bottlenecks, while the fixed-last restriction is motivated by the fact that the makespan is defined on M_m (but excludes interior pairs (k, l) with $l \neq m$).

Table 1 summarizes the most common restrictions discussed above.

3.3 Strategy design and automation

Beyond manually designed heuristics, recent studies have explored automated approaches to construct B&B components. For example, Maudet and Danoy (2025) and Vorokhta et al. (2026) investigate the use of genetic programming to generate or select effective strategies in B&B, showing that the policy design itself can be optimized. More broadly, the metaheuristics literature emphasizes that algorithmic performance is often driven by a small number of reusable building blocks and design decisions (e.g., scoring rules, selection policies, update schedules), which makes automatic design and tuning attractive when these choices are instance-dependent (Talbi (2009)).

3.4 Parallel implementations

From a systems perspective, PFSP is also served as a benchmark for multicore and GPU-based B&B implementations, as shown by Melab et al. (2012); Mezmaz et al. (2013); Helbecque et al. (2023). These works demonstrate that reducing the work per node and efficiently managing memory access are crucial to performance. They also highlight additional constraints that do not appear in sequential solvers, such as load balancing across threads and the need to keep per-node computations predictable. In particular, early-exit behavior and adaptive pair ordering can introduce runtime variability that interacts with scheduling and work stealing.

Although previous studies have explored reduced machine-pair sets, adaptive evaluation orders, automated strategy generation, and parallel execution, the practical cost of the pair-selection policy itself has received much less attention.

In the following section, we focus on this aspect and study several lightweight variants of machine-pair selection policies for LB_2, focusing on their interaction with frequent bound evaluations and early-exit pruning within PFSP B&B.

4 Proposed machine-pair selection strategies

This section introduces the LB_2 machine-pair selection baselines developed in this work and used in our experiments. We consider a total of ten additional baselines: two fixed (static) strategies and eight learning-based variants. All methods rely on the same underlying computation: for a given pair of machines (M_k, M_l) we solve the corresponding two-machine relaxation using Johnson-type ordering.

The variants differ only in algorithmic choices: which machine pairs are evaluated, in what order

they are considered, and how this order is updated during the search. We restrict attention to policies that modify only the evaluation sequence while preserving the asymptotic per-node complexity of the baseline evaluation strategy. This distinction is important because PFSP B&B methods may explore large search trees, and even modest increases in per-node complexity can offset the gains obtained from reducing the number of evaluated pairs.

In the solver, pairs are evaluated sequentially and the evaluation stops as soon as $LB \geq UB$, so good ordering can reduce computational cost even when the underlying set W remains unchanged. When many nodes are close to being pruned, early encountering a strong pair can turn a nearly complete pass over W into only a few evaluations, while a loose upper bound rarely triggers early exit and makes pair selection mainly affect the final bound value rather than runtime.

Restricting the set of evaluated pairs is therefore important. Computing a single two-machine relaxation is inexpensive, but doing so $\binom{m}{2}$ times at every node quickly becomes costly. Over large search trees, these small costs accumulate and can end up dominating runtime. The baselines considered here aim to learn which pairs tend to be most useful during the search.

We now introduce the baselines. Each can be characterized by the following features:

- pair ranking (learned / random),
- number of pairs selected (k) with multiple choices examined and given as an extension of the name for several variants,
- frequency of update (none / periodic / per node),
- additional control parameters such as α, a, the smoothing factor used in the exponentially weighted moving average (EWMA) update rule, as well as mechanisms such as tie-sharing, shrinking, and adaptive selection.

All variants considered in this paper compute $LB_2(W)$ for a particular (possibly dynamic) subset W of machine pairs.

4.1 Random static subsets (fixed for the whole run) — *RS*

As a simple baseline, we also evaluate random static subsets of machine pairs. For a given subset size $k \in \{1,\ \lfloor m/2 \rfloor + 1,\ m\}$, the subset is sampled once at the beginning of the run and then kept fixed throughout the search, giving $W = W^{\text{rand}}(k) \subseteq W_0$.

This variant serves as a reference point for understanding how much of the observed performance gains come from informed pair selection rather than simply reducing the number of evaluated pairs.

4.2 Random reshuffling (training-length cycles) — *RR*

We also consider a random-with-resets strategy. Instead of keeping a single random subset for the entire run, the solver periodically discards the current subset and samples a new one of the same size.

We define the reshuffling period as follows: each cycle begins with a full-pass "training" phase (evaluating all pairs), followed by an exploitation phase that uses a randomly sampled active set of size $k \in 1,\ \lfloor m/2 \rfloor + 1,\ m$. At the next reset, the active set is discarded and resampled, so that $W = W_c^{\text{reset}}(k) \subseteq W_0$ can change from cycle to cycle.

4.3 Adaptive top-k evaluation of LB_2 — *AT*

We follow and extend the general online-learning idea briefly introduced in the work of Gmys et al. (2020) (maintaining an active machine-pair set with periodic full refresh). Building on this, this subsection summarizes the mechanism and the parameters we use.

Beyond static restrictions, adaptive policies aim to reduce the average number of evaluated pairs per node by exploiting early exit and prioritizing pairs that tend to be most informative during the search. In the paper of Gmys et al. (2020), we introduced a lightweight online mechanism that maintains an active set $W^\star \subset W_0$ of promising pairs and first evaluates them, and the set is dynamically updated based on the usefulness observed. We parameterize this learning schedule by:

- the active-set size k fixed and defined at the beginning,

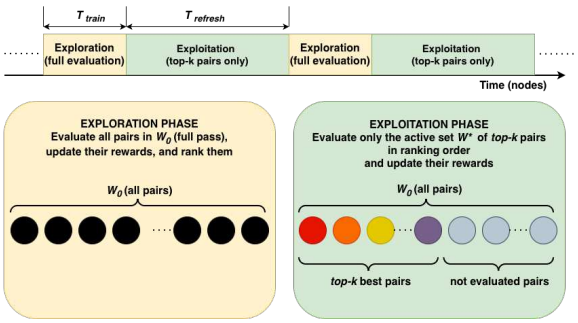


Fig. 2 Schematic overview of the adaptive top-k learning mechanism.

- a training window length T_{train} (number of bound computations during which a full pass is performed to (re)estimate pair utility),
- a full-refresh interval T_{refresh} (number of bound computations between two full passes).

In the configuration reported by Gmys et al. (2020), we used $k = m$, $T_{\text{train}} = 2n$ and $T_{\text{refresh}} = 200n$, where n is the number of jobs (Gmys et al. (2020)). Based on preliminary analysis, we additionally include $k = m/2 + 1$ in the experiments.

In this approach, a pair is considered "successful" if it frequently triggers pruning by being the first pair to reach $LB \geq UB$ (a winner-take-all criterion). Operationally, pairs are maintained in a list sorted by their rewards and are evaluated in that order.

Figure 2 provides a schematic overview of the adaptive top-k learning mechanism, highlighting the alternation between exploration (full pair evaluation) and exploitation (evaluation of the active top-k set).

4.4 Adaptive evaluation with tie-shared rewards — *TA*

The original adaptive scheme credits a single "winner" pair, although ties are common, as several pairs can produce the same best value and early exit may make multiple pairs equally responsible for pruning. To reduce the sensitivity to arbitrary tie-breaking, we share the reward across all tied-best pairs. The remainder of the method follows the adaptive scheme described in Subsection 4.3.

Let W be the set of eligible pairs at a node. Since several pairs can achieve the same best score, we first compute the best value M and then collect all pairs that achieve this value into the tie-best set T (so the links are handled explicitly, rather than arbitrarily broken).

$$M = \max_{(p,q)\in W_0} LB_{2_{p,q}},$$
$$T = \{(p,q) \in W_0 : LB^{(i)}_{2_{p,q}} = M\}. \tag{5}$$

We assign

$$r_{i,j} = \begin{cases} 1/|T|, & (i,j) \in T, \\ 0, & \text{otherwise.} \end{cases} \tag{6}$$

4.5 Learning restricted to extreme-machine pairs with tie-shared rewards — *Ext-TA*

Empirical motivation

To motivate this restriction, we first ran a small pre-analysis using the classical full evaluation $LB_2(W_0)$. For each bound computation, we recorded the single machine pair (M_i, M_j) that achieved the maximum two-machine value.

Figures 3–4 show, for two representative instances, how often each pair was selected as the best in a 10-minute run. In both cases, the highest counts concentrate on pairs that involve the first machine ($i = 1$, first row) or the last machine ($j = m$, last column).

Extreme pool and learning rule

Motivated by this observation, we reduce the learning search space by restricting candidates to extreme pairs,

$$W_{\text{ext}} = \{(1,j) : j = 2,\ldots,m\} \cup \{(i,m) : i = 1,\ldots,m-1\}, \tag{7}$$

and run the learning mechanism within this pool, using the shared-reward update of Section 4.4 and following the adaptive scheme from Subsection 4.3.

4.6 Adaptive evaluation with alternative tie-shared scoring — *SB-TA*

This variant keeps the same overall mechanism (an active set of k pairs that is periodically refreshed), but replaces the winner-based reward signal with a complex score computed from the observed LB_2

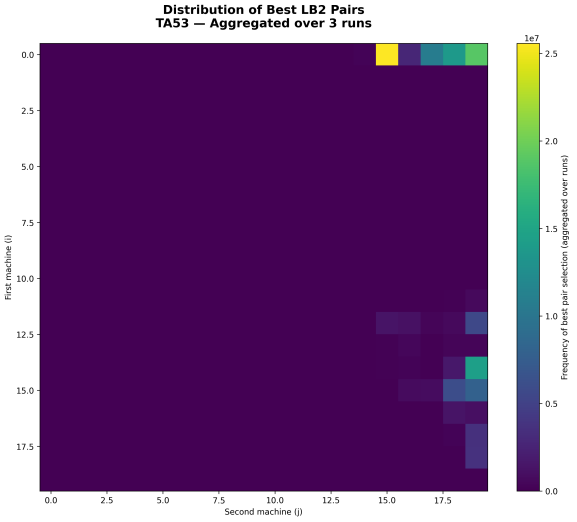


Fig. 3 Best-pair frequency heat map for a representative Taillard PFSP instance (10-minute run; full $LB_2(W_0)$; winner-takes-all). Dark purple indicates zero selections; yellow indicates the maximum frequency observed on these runs.

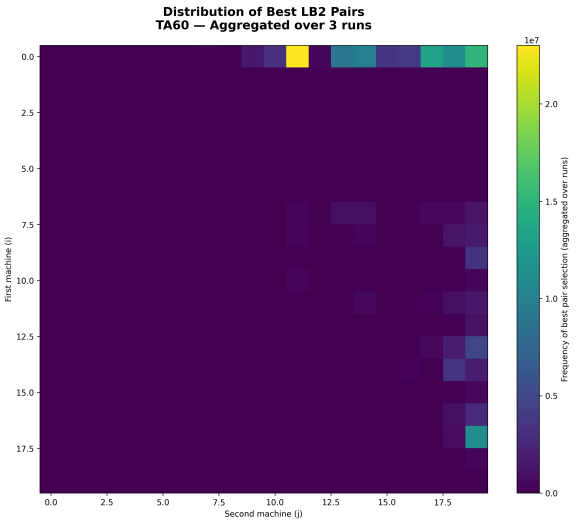


Fig. 4 Best-pair frequency heat map for a second representative Taillard PFSP instance under the same protocol as Fig. 3.

values. We form the next active set $W^{\star}_{\text{score,tie}}(k)$ by selecting the top-k pairs under this score.

Let the scoring window contain N observations indexed by $r \in \{1, \ldots, N\}$. For each r and each pair $(i, j) \in W_0$, we compute $LB2^{(r)}_{i,j}$ and its average

$$\text{meanLB}_{i,j} = \frac{1}{N} \sum_{r=1}^{N} LB2^{(r)}_{i,j}. \tag{8}$$

Let T denote the tied-best set defined in Section 4.4. We compute a tie-shared best-frequency

$$f_{i,j} = \frac{1}{N} \sum_{r=1}^{N} \mathbf{1}[(i,j) \in T] \cdot \frac{1}{|T|}, \tag{9}$$

and define the composite score

$$\text{score}_{i,j} = \text{meanLB}_{i,j}\,(1 + f_{i,j}). \tag{10}$$

Intuitively, $f_{i,j}$ measures how often a pair appears among the tied-best candidates, averaged over N observations and normalized so that each tie contributes its credit uniformly across all tied pairs. This avoids over-rewarding nodes where many pairs share the same best value. The final score then combines the average quality of a pair, $\text{meanLB}_{i,j}$, with its stability as a frequent tied-best choice, so that pairs that are both strong and consistently competitive among ties receive higher scores.

The active set is then formed by selecting the k pairs with the largest scores.

4.7 EWMA memory in adaptive evaluation — *EWMA*

This variant also follows the same general framework as the adaptive evaluation mechanism (the same refresh schedule and the active set of size k), but replaces score resets with a smoothed memory of past rewards.

Let $s_{i,j}(t)$ denote the score of pair (i, j) after the t-th update (bound computation), and let $r_{i,j}(t)$ denote the reward observed at that update using the original winner-take-all credit assignment (that is, a single "winner" pair receives reward 1 and all others receive 0). We maintain an EWMA score Hunter (1986)

$$s_{i,j}(t) = \alpha\, s_{i,j}(t-1) + (1-\alpha)\, r_{i,j}(t), \qquad \alpha \in (0,1). \tag{11}$$

At each refresh time in the adaptive evaluation schedule, the next active set $W^{\star}_{\mathrm{ewma}}(k,\alpha)$ is formed by selecting the k pairs with the largest current scores $s_{i,j}(t)$, and the pairs are evaluated in decreasing-score order.

4.8 Championship-based competitive pairs — *Champ*

We also consider a simple "championship" selection rule, inspired by dueling bandits, in which pairs compete for points and are always kept in a global ranking. We use an elite size k and a championship size of 1 (one duel per node after the champion test).

Let $s_i(t)$ be the current score of the pair i after the t-th update. At any time, all pairs are sorted in decreasing order of $s_i(t)$. At a node, we first evaluate the current best-ranked pair (the *champion*). If this already yields $LB_2 > UB$, the node is pruned and the evaluation stops.

Otherwise, we perform one duel by sampling two pairs j and k uniformly at random and evaluating them (ensuring that the champion is not re-tested if it is selected again). The duel reward (Δ_j, Δ_k) is defined as follows: if both evaluations exceed the UB, then $\Delta_j = \Delta_k = 0.5$; otherwise, the better of the two receives $\Delta = 1$ and the other receives 0. If at least one of the two evaluations exceeds the UB, the node is pruned.

Finally, we update the two selected scores,

$$\begin{aligned} s_j(t) &= s_j(t) + \Delta_j, \\ s_k(t) &= s_k(t) + \Delta_k, \end{aligned} \tag{12}$$

and re-sort the pairs accordingly.

4.9 Shrinking method in adaptive evaluation — *Shrink*

This variant keeps the same refresh schedule and the same learning/update rule as in Subsection 4.3, but makes the active-set size k adaptive within each exploitation period.

At the beginning of every exploitation period (immediately after the full-pass training phase), we initialize the active-set size to the original choice: $k = m$. We also enforce the same minimum active-set size, namely $k_{\min} = \lfloor m/2 \rfloor + 1$.

Let $u(t)$ denote the number of pairs that are actually evaluated at node t before the early exit stops the evaluation. At the next node, if the previous evaluation used less than half of the current active set, $u(t-1) < \frac{1}{2}k$, we shrink the budget by one,

$$k = \max\{k_{\min},\, k-1\}, \tag{13}$$

and proceed exactly as in the standard adaptive top-k variant from Subsection 4.3 with this updated k (i.e., keep maintaining the reward-sorted list and evaluate the current top-k pairs first).

After the next full refresh/reset, the procedure restarts from $k = m$.

4.10 Knee-selected budget in adaptive evaluation — *Knee*

This variant again follows the online refresh schedule and the learning rule from the Subsection 4.3, but selects the active-set size k automatically at the end of each training phase.

Let $k_{\min} = \lfloor m/2 \rfloor + 1$ and $k_{\max} = 2m$. After the full-pass training phase, each pair (i,j) has an accumulated training score $R_{i,j}$, i.e., the cumulative reward from the same update rule used above (winner-takes-all). We sort all pairs in decreasing $R_{i,j}$. Let $v_1 \geq v_2 \geq \cdots \geq v_{k_{\max}}$ denote the resulting top-$k_{\max}$ list of scores.

We then normalize these values by their total mass

$$S = \sum_{t=1}^{k_{\max}} v_t, \tag{14}$$

and define the normalized contribution

$$p(t) = \frac{v_t}{S}, \qquad t = 1, \ldots, k_{\max}. \tag{15}$$

If $S < 10^{-9}$, we set $k = k_{\min}$. Otherwise, we interpret the "knee" as the point where the marginal normalized contribution drops the most, i.e., we select

$$t^{\star} \in \arg\max_{t \in \{k_{\min}, \ldots, k_{\max}\}} \big(p(t-1) - p(t)\big), \tag{16}$$

(with the convention $p(0) = 0$), and set the exploitation budget to

$$k = \min\{k_{\max},\, \max\{k_{\min},\, t^{\star}\}\}. \tag{17}$$

Intuitively, this picks the prefix length at which the next candidate pair becomes negligible under

the training-phase reward distribution. When this happens very early, the minimum-budget safeguard $k \geq k_{\min}$ prevents selecting an overly small active set.

After the next full refresh/reset, the knee selection is repeated.

4.11 Machine-pair selection configurations

Table 2 summarizes the considered strategies and their main characteristics.

Table 2 Summary of machine-pair selection strategies.

Baseline	Number of pairs	Update frequency	Source
AP	$\binom{m}{2}$	—	Prior
Adj	$m-1$	—	Prior
Last	$m-1$	—	Prior
RS	k	—	Paper
RR	k	periodic	Paper
AT	k	periodic	Paper
TA	k	periodic	Paper
Ext-TA	k	periodic	Paper
SB-TA	k	periodic	Paper
EWMA	k	periodic	Paper
Champ	all	per node	Paper
Shrink	adaptive	periodic	Paper
Knee	adaptive	periodic	Paper

5 Experiments

This section describes the experimental setup used to evaluate the practical impact of machine-pair selection for LB_2 within the B&B framework. For completeness, we also report results for the classical one-machine bound LB_1 as a cheap baseline. We first introduce the instance sets and the protocol used to generate comparable subproblems and then describe the solver configuration and evaluation metrics.

5.1 Instance sets

We consider only instances from the Taillard PFSP benchmark (Taillard (1993)). Since our goal is to compare the bounding policies inside B&B, we focus on the cases where the choice of pair-selection strategy has a noticeable effect within a fixed time budget.

We first examine medium-size instances with at least 10 machines. To select representative cases, we ran the solver with full evaluation $LB_2(W_0)$ and kept instances whose baseline runtime lies roughly between 10 minutes and the global time limit of 3 hours. This yields the subset ta11–ta16, ta18, ta20, and ta22 (for example, ta20 corresponds to 20×10 and ta22 to 20×20). For these instances, the solver typically proves optimality within the time limit, which allows us to interpret runtime differences directly in terms of how pair selection affects pruning efficiency and per-node work.

To include larger instances while keeping the comparison fair, we also consider ta51–ta60 (50×20). These instances are not solved to optimality within the given time limit. To make the comparison more consistent, we evaluate all methods with respect to the same reference point in the search process.

The protocol is as follows:

- first, we run the B&B algorithm with the full $LB_2(W_0)$ configuration for one hour;
- after one hour, we record how many nodes have been processed up to this point, including both explicitly explored nodes and nodes implicitly discarded by pruning;
- this one-hour state is then fixed as a common reference point;
- for every other configuration, we measure the time required to reach this same state, i.e., to process the same number of nodes and reproduce the recorded search state.

This protocol improves reproducibility, since all configurations start from exactly the same pool of subproblems instead of potentially different trajectories.

The initial UB is initialized to the largest possible integer value ("infinity"). Thus, the search starts without any incumbent solution, eliminating the influence of heuristic warm starts on the results.

5.2 Sensitivity analysis

Before running the main comparisons, we performed a small sensitivity study on some Taillard instances. Using the parameterization introduced above, we varied:

- the active-set size k (top-k pairs),
- the length of the training window T_{train} ($T_{\text{train}} = 2n$ is used by Gmys et al. (2020)),
- the full-refresh interval T_{refresh} ($T_{\text{refresh}} = 200n$) is used by Gmys et al. (2020),

where n is the number of jobs.

In all instances tested, we observed that the parameter k has the largest impact, while T_{train} and T_{refresh} have a relatively minor effect within the tested ranges. In the main learning-based experiments, we therefore use $k \in \{m,\ \lfloor m/2 \rfloor + 1\}$.

Representative sensitivity results are summarized in Tables 3–4. In both tables, we use the original adaptive evaluation method, varying only the active-set size (top-k pairs). A clear "knee" appears around half of the available pairs: once k exceeds this midpoint, performance improves sharply—either through a noticeable runtime reduction (Table 3) or through a significantly better solution within a fixed time budget (Table 4). The same pattern was observed across several sizes of Taillard instances.

Table 3 Sensitivity to top-k on Taillard instance `ta14` ($m = 10$, $n = 20$)

top-k	runtime (s)
1	280.0
2	67.6
3	36.2
4	28.8
5	25.0
6	23.4
7	22.5
8	21.8
9	21.6
10	21.3

Table 4 Sensitivity to top-k on Taillard instance `ta27` ($m = 20$, $n = 20$)

top-k	best value (10 min)
1	2428
2	2413
3	2400
4	2390
5	2390
6	2390
7	2384
8	2384
9	2379
10	2379
11	2379
12	2379
13	2379
14	2379
15	2379
16	2379
17	2379
18	2379
19	2379
20	2379

5.3 B&B configuration

We ran a sequential (single-thread) B&B so that the search remains deterministic and the comparisons are reproducible. Each evaluation is repeated three times, and we report the mean and standard deviation.

The node selection follows a depth-first search (DFS) strategy described by Helbecque et al. (2023). DFS is well suited in this context, as it makes per-node overhead more visible. As the solver goes deeper in the tree, the bound is repeatedly evaluated on closely related subproblems. This also helps evaluate the consistency of a pair-selection policy and whether the time spent selecting machine pairs is justified.

Branching is performed forward on the permutation: at depth d, the decision fixes the job placed in position $d+1$. Each node corresponds to a subproblem with a fixed prefix and an unfixed remainder. As the set of unscheduled jobs shrinks, the relative importance of machine pairs may change. In such settings, adaptive policies are more effective than static restrictions, whereas overly aggressive restrictions can reduce the amount of information captured from potential bottlenecks.

5.4 Empirical motivation for pair reduction

To further motivate the reduction of evaluated machine pairs in LB_2, we performed a preliminary analysis of the full evaluation scheme $LB_2(W_0)$, in which all pairs of machines $\binom{m}{2}$ are considered at each node.

Although computing a single two-machine relaxation is relatively inexpensive, the cumulative cost of evaluating all pairs becomes significant when repeated over the search tree. To better

understand this effect, we measured the average per-node time spent on the shared part of the LB_2 computation (i.e., preprocessing that is reused across all machine pairs) and compared it to the average time spent on evaluating all machine pairs.

For this experiment, a simple baseline B&B solver in C with the full LB_2 evaluation was implemented. The search procedure constructed partial permutations in a straightforward left-to-right manner, starting from an empty schedule, and used an initial upper bound set to the maximum possible integer value. To ensure comparability between instance sizes while keeping the experiment lightweight, the number of explored nodes was limited to 10^7.

We performed this analysis on Taillard instances of varying sizes, with the number of machines ranging from 10 to 20 and the number of jobs ranging from 20 to 500. All experiments were performed on a node of the Aion HPC cluster at the University of Luxembourg, equipped with 128 CPU cores and 256 GB of RAM.

Table 5 summarizes the average per-node times across different instance sizes.

The results consistently show that machine-pair evaluations dominate the computational cost, despite the fact that the shared preprocessing component is only computed once per node.

5.5 Implementation and execution environment

All experiments were performed using the Chapel-based PFSP B&B solver of Helbecque et al. (2024). The runs were carried out on the University of Luxembourg HPC platform, as described in Varrette et al. (2014), using one node with 128 CPU cores and 256 GB of RAM.

As discussed in Section 2, we also used the optimized incremental evaluation of LB_1: the head/tail terms are computed once per parent node in $O(mn)$ time and the bound is updated for each of its $O(n)$ children in $O(m)$ time (overall $O(mn)$ for the whole child pool).

To keep the comparison clean, we report sequential results (single-thread runs). This avoids potential confounding effects from parallel scheduling, synchronization overhead, or memory locality. Each run is limited to 3 hours of wall-clock time.

This setup allowed us to focus on the effect of the pair-selection policies themselves. In parallel environments, additional factors, such as cache behavior or resource contention, may affect performance, so we leave the analysis of parallel settings for future work.

The source code of the proposed methods and experimental framework is publicly available at: https://gitlab.com/uniluxembourg/snt/pcog/ultrabo/p3d-dfs-pair-selection.

5.6 Performance metrics

We primarily report wall-clock time. For each instance, we record either the time to termination (when optimality is proven) or the time to-limit (3 hours).

5.7 Results and discussion

To make the comparison easier to read, Table 6 reports three configurations for each instance: the full-evaluation baseline, the static *Last* restriction, and the best-performing adaptive policy. Runtime ratios are also included to support direct comparison.

For completeness, Table A1 reports mean sequential B&B runtime in seconds for all configurations. Smaller values indicate better performance. We use heat-map coloring to support comparison: the best value in each column is shown in blue, and the remaining values follow a red-to-green scale (greener is better). The scale saturates at 2× the best value, and values beyond this threshold remain red. Table A2 reports normalized means, and Table A3 reports standard deviations.

We can observe several consistent patterns. First, a weak bound (LB_1) frequently reaches the time limit. Second, evaluating all machine pairs (LB_2 AP, i.e. $W = W_0$) is almost never optimal. Despite producing the strongest per-node bound, it is systematically dominated by cheaper strategies: the additional pruning power cannot offset the quadratic evaluation cost that grows with $|W_0|$, making each node expansion disproportionately expensive. Similarly, simple structured static subsets such as *Adj* are not robust and often time out.

Among static restrictions, anchoring pairs on the last machine (LB_2 *Last*) provides a strong baseline. It yields substantial speedups over AP

Table 5 Average per-node time comparison for full $LB_2(W_0)$ evaluation (times reported in microseconds).

Instance	Shared preprocessing time, us	Pair evaluation time, us
ta14 (n=20, m=10)	0.184	3.200
ta24 (n=20, m=20)	0.318	16.252
ta44 (n=50, m=10)	0.483	4.327
ta54 (n=50, m=20)	0.847	28.933
ta74 (n=100, m=10)	0.953	7.847
ta84 (n=100, m=20)	1.928	34.736
ta94 (n=200, m=10)	1.778	19.040
ta104 (n=200, m=20)	3.908	56.855
ta114 (n=500, m=20)	9.941	126.140

on smaller instances and remains competitive on several larger ones. However, its performance is uneven, with Ta55 reaching the time limit, suggesting that a fixed subset cannot fully capture the diversity of informative pairs across the benchmark.

The most reliable results are obtained by adaptive policies that explicitly trade off bound strength against evaluation cost. Across many columns, *AT / TA / Ext-TA* and *EWMA* variants rarely exhibit catastrophic failures. Ext-TA with full budget is often one of the best-performing proposed methods and is the best in several instances, while its smaller-budget setting is less stable in some cases.

Looking at the results, we can see that the choice of k matters. Larger budgets ($k = m$) are beneficial on some harder instances. For example, Ta18 is solved by *AT / TA* with $k = m$ but not with the smaller $k = m/2 + 1$ setting. At the same time, smaller budgets can be preferable when the evaluation cost dominates: several Ta5x instances favor $k = m/2+1$. The *Shrink* and *Knee* heuristics behave as robust automatic choices, typically matching the best fixed-budget AT variants without requiring a manual selection of k.

In contrast, methods with a large random component (*RS/RR*) exhibit high variance and frequent timeouts. The championship-style competitive update (*Champ*) is generally weaker than the best adaptive policies, but can still be competitive on some smaller instances.

In general, no single configuration is uniformly the best across all instances. Nevertheless, the findings show that the safest default is to use an adaptive method. This includes *AT, TA,* and *EWMA* variants, as well as *Shrink* and *Knee* when automatic budget control is preferred. These methods consistently achieve runtimes close to the best observed values, while avoiding the severe slowdowns that appear with full evaluation, fixed weak subsets, or randomized selection rules.

6 Conclusion and future work

In this work, we investigated how the choice of machine pairs affects both the computational cost and the effectiveness of the two-machine bound LB_2 within an exact PFSP B&B algorithm. In addition to classical static restrictions, we explored several adaptive selection strategies based on online rewards, including tie-aware updates, extreme-machine restrictions, and smoothed scoring mechanisms. Our experiments also show that dynamically adapting the active-set size k can play an important role in overall performance.

The experiments on the Taillard benchmark indicate that evaluating all $\binom{m}{2}$ machine pairs ($LB_2(W_0)$) is usually not the most efficient option in terms of total runtime. In many cases, cheaper selection policies perform better because they reduce the computational effort at each node while still being able to identify useful pairs early enough to improve pruning. These results indicate that when a bound is computed at every node of the search tree, the focus should not be only on maximizing the quality of the bound. Rather, it is important to balance the pruning power with the computational cost.

Among the static approaches, the *Last* restriction provides a strong and simple baseline, although its behavior remains highly instance-dependent. In contrast, adaptive strategies (*AT /*

Table 6 Filtered results. We report the LB_2 full evaluation baseline, the static *Last*-pair restriction, and the best mean runtime among the proposed adaptive policies. Ratios are reported relative to LB_2 Full. All runtimes are in seconds (s).

Instance	LB_2 ***Full***	LB_2 ***Last***	***Last*/Full**	**Best method**	**Best (s)**	**Best/*Full***
Ta11	35.1	24.7	0.70	*AT* (k=m)	14.8	0.42
Ta12	97.4	59.8	0.61	*Ext-TA* (k=m)	51.3	0.53
Ta13	476.2	338.3	0.71	*AT* (k=m)	268.8	0.56
Ta14	39.3	21.7	0.55	*AT* (k=m)	18.8	0.48
Ta15	127.9	61.2	0.48	*TA/Ext-TA* (k=m)	57.1	0.45
Ta16	2202	1296	0.59	*Shrink*	845	0.38
Ta18	4355	2656	0.61	*Ext-TA* (k=m)	2406	0.55
Ta20	1133	651	0.57	*Shrink*	600	0.53
Ta22	4215	802	0.19	*Ext-TA* (k=m)	684	0.16
Ta51	3766	705	0.19	*AT* (k=m/2+1)	536	0.14
Ta52	3714	732	0.20	*Ext-TA* (k=m)	493	0.13
Ta53	3710	763	0.21	*Ext-TA* (k=m)	529	0.14
Ta54	3617	814	0.22	*Shrink*	554	0.15
Ta55	4278	10800	2.53	*AT* (k=m/2+1)	658	0.15
Ta56	3664	2143	0.58	*AT* (k=m/2+1)	518	0.14
Ta57	3699	1045	0.28	*Shrink*	545	0.15
Ta58	3638	961	0.26	*Knee*	458	0.13
Ta59	3690	1049	0.28	*AT* (k=m/2+1)	459	0.12
Ta60	3636	1025	0.28	*Knee*	669	0.18

TA / EWMA / Ext-TA, together with automatic *Shrink / Knee* budget mechanisms) tend to perform more consistently across different instances (Table 6). Finally, while the *Champ* update is not among the top performers overall, it remains a simple option that can still yield competitive results on some smaller instances.

More generally, our results suggest that learning which pair evaluations are most informative can itself be viewed as part of the bounding procedure rather than as a purely implementation-level optimization. Although this study focuses on LB_2, the same idea could naturally be applied to other decomposable bounds where many candidate subcomputations compete for a limited computational budget.

Several directions remain open for future work:

- designing hybrid strategies that adapt the budget during the search, for example, by using very cheap screening policies near the root and more selective adaptive budgets deeper in the tree,
- developing more robust learning signals and stabilization mechanisms, such as depth-aware rewards, variance reduction techniques, or safeguards against overfitting to a small subset of pairs,
- studying how adaptive pair selection interacts with other B&B components, including branching strategies, node selection policies, and dominance rules.

Declarations

Funding

This work is funded by the Luxembourg National Research Fund (FNR), INTER programme under the UltraBO Project (ref. INTER/ANR/22/17133848).

Conflict of interest

The authors have no competing interests to declare that are relevant to the content of this article.

Ethics approval and consent to participate

Not applicable.

Consent for publication

Not applicable.

Data availability

Not applicable.

Material availability

Not applicable.

Code availability

Not applicable.

Author contribution

All authors contributed to the study conception and design. Material preparation, data collection, and analysis were performed by Alisa Vorokhta, Jan Gmys and Gwen Maudet. The first draft of the manuscript was written by Alisa Vorokhta and all authors commented on previous versions of the manuscript. All authors read and approved the final manuscript.

Appendix A Supplementary Experimental Results

For completeness, we include three additional tables. Table A1 reports mean runtimes and uses the same heat-map coloring as in the main text. It also includes a *Wins* column, which counts how often each method achieves the best (lowest) runtime, with ties counted as shared wins. Table A2 reports normalized mean runtimes relative to LB_2 AP. Table A3 reports the corresponding standard deviations of runtime.

Table A1 Mean runtime (seconds) on Taillard PFSP instances (time limit: 3 hours; unsolved runs are reported as 10800 s). The Wins column counts in how many instances the corresponding method achieved the best (lowest) runtime among all compared configurations.

Method	Ta11	Ta12	Ta13	Ta14	Ta15	Ta16	Ta18	Ta20	Ta22	Ta51	Ta52	Ta53	Ta54	Ta55	Ta56	Ta57	Ta58	Ta59	Ta60	Wins
LB_1	117.2	172.5	472.9	154.1	490.2	10800.0	10257.4	10800.0	9998.0	10800.0	10800.0	6951.0	8519.3	10800.0	10800.0	10800.0	10800.0	10800.0	3417.7	**0**
LB_2 *AP*	35.1	97.4	476.2	39.3	127.9	2202.3	4354.7	1133.2	4214.8	3765.8	3713.8	3709.9	3616.5	4277.7	3663.7	3699.3	3638.3	3690.1	3635.6	**0**
LB_2 *Adj*	151.0	218.1	681.1	147.9	632.6	10800.0	10800.0	10800.0	10800.0	10800.0	10800.0	10800.0	10800.0	10800.0	10800.0	10800.0	10800.0	10800.0	9858.7	**0**
LB_2 *Last*	24.7	59.8	338.3	21.7	61.2	1295.8	2656.4	650.9	801.7	705.1	732.4	763.0	813.5	10800	2142.9	1044.6	961.1	1049.0	1025.0	**0**
LB_2 *AT* ($k = m/2+1$)	15.1	63.3	286.8	21.8	61.1	878.8	10800	1545.8	697.8	536.0	494.5	532.2	559.6	657.9	517.8	643.9	459.3	459.3	687.3	**4**
LB_2 *AT* ($k = m$)	14.8	55.9	268.8	18.8	59.6	872.4	2509.9	612.1	712.0	619.6	539.9	572.9	566.5	720.4	529.3	563.9	477.4	491.8	676.3	**3**
LB_2 *TA* ($k = m/2+1$)	26.6	92.1	311.7	19.6	178.1	2089.5	10800	630.3	2086.1	701.1	507.5	1599.4	10800	10800	2560.5	10800	687.8	476.4	1490.4	**1**
LB_2 *TA* ($k = m$)	15.9	56.7	277.7	19.0	57.1	916.6	2471.0	618.3	732.8	624.1	545.8	578.6	622.7	739.5	552.3	711.7	492.2	498.1	1100.0	**1**
LB_2 *Ext-TA* ($k = m/2+1$)	28.1	6128	308.5	520.2	215.9	10800	10800	1546.5	5716.5	10800.0	526.8	10800.0	10800.0	10800.0	10800.0	10800.0	10800.0	629.2	654.1	**1**
LB_2 *Ext-TA* ($k = m$)	21.6	51.3	293.1	19.7	57.1	1097.8	2406.0	623.8	683.9	567.8	493.2	528.7	574.2	10800.0	1525.0	686.4	621.8	663.0	678.3	**5**
LB_2 *SB-TA* ($k = m/2+1$)	17.8	66.0	349.3	23.8	67.2	1041.3	4391.7	722.2	999.5	630.6	600.5	730.0	736.1	10800.0	1527.8	718.2	659.5	832.6	1043.2	**0**
LB_2 *SB-TA* ($k = m$)	16.7	59.3	324.6	23.0	65.0	1040.5	3513.1	627.5	972.4	720.4	644.3	758.5	725.9	918.8	1070.1	755.3	647.2	846.0	1025.5	**0**
LB_2 *RS* ($k = 1$)	10800.0	10800.0	10800.0	10800.0	10800.0	10800.0	10800.0	10800.0	10800.0	10800.0	10800.0	10800.0	10800.0	10800.0	10800.0	10800.0	10800.0	10800.0	10800.0	**0**
LB_2 *RS* ($k = m/2+1$)	79.0	3800.4	4580.8	4298.5	10800.0	10425.7	10800.0	8383.9	8401.2	7412.6	10800.0	10800.0	8084.9	10800.0	10800.0	10800.0	9498.3	10800.0	9450.7	**0**
LB_2 *RS* ($k = m$)	3620.9	133.0	416.6	58.7	196.6	6873.9	8264.5	9026.2	7326.9	10800.0	10800.0	8242.4	8034.7	10800.0	6111.5	4121.1	4003.7	5775.2	3639.8	**0**
LB_2 *RR* ($k = 1$)	1600.8	5475.0	10800	2284.8	5368.4	10800.0	10800.0	10800.0	10800.0	10334.3	10800.0	10800.0	10800.0	10800.0	10800.0	10800.0	10800.0	10800.0	10800.0	**0**
LB_2 *RR* ($k = m/2+1$)	72.7	219.0	1022.8	101.5	327.3	6625.3	10800.0	5199.4	7799.4	5347.3	9619.5	6975.9	9472.9	10800.0	10124.4	8128.8	10800.0	9744.2	6406.6	**0**
LB_2 *RR* ($k = m$)	44.7	130.2	564.3	59.2	195.5	3697.3	7709.7	2758.0	4877.5	3479.8	7609.9	5571.7	10800.0	10800.0	6111.5	4121.1	9077.1	5775.2	3639.8	**0**
LB_2 *EWMA* ($k = m/2+1,\ \alpha = 0.05$)	15.3	64.1	290.7	22.2	62.1	893.7	10800.0	1567.0	722.1	544.4	509.1	547.9	580.5	676.4	534.5	662.3	474.0	474.6	707.2	**0**
LB_2 *EWMA* ($k = m/2+1,\ \alpha = 0.1$)	15.4	64.7	292.7	22.3	62.5	901.2	10800.0	1582.9	724.9	545.9	510.6	548.3	579.5	678.1	536.2	663.4	473.1	474.1	710.6	**0**
LB_2 *EWMA* ($k = m,\ \alpha = 0.05$)	15.1	56.8	275.5	19.1	60.5	888.7	2560.9	619.9	734.7	627.9	554.4	590.0	585.5	738.4	543.0	579.9	492.0	505.6	694.8	**0**
LB_2 *EWMA* ($k = m,\ \alpha = 0.1$)	15.2	57.3	274.4	19.3	61.0	894.4	2582.9	625.2	738.2	630.8	556.1	588.7	585.2	739.4	546.1	582.9	492.4	505.9	696.7	**0**
LB_2 *Champ* ($k = m/2+1$)	19.3	97.4	367.0	34.2	90.4	1043.6	3837.1	694.6	862.1	10800.0	10800.0	10800.0	10800.0	10800.0	3273.6	10800.0	10800.0	2708.9	886.5	**0**
LB_2 *Champ* ($k = m$)	17.8	92.9	350.6	33.2	80.9	1076.2	2979.3	596.7	876.6	10800.0	10800.0	10800.0	10800.0	10800.0	2374.7	10800.0	10800.0	2215.4	878.7	**1**
LB_2 *Shrink*	15.1	61.2	298.2	19.1	59.8	845.3	3114.2	599.9	691.8	595.1	500.7	539.9	553.5	669.9	522.5	545.1	458.5	463.6	690.2	**3**
LB_2 *Knee*	14.8	60.1	286.9	21.9	60.4	868.2	3114.2	960.3	698.1	544.7	495.6	532.5	565.9	666.7	549.0	556.9	458.2	460.3	668.7	**2**

Table A2 Normalized mean runtime (relative to LB_2 AP) on Taillard PFSP instances.

Method	Ta11	Ta12	Ta13	Ta14	Ta15	Ta16	Ta18	Ta20	Ta22	Ta51	Ta52	Ta53	Ta54	Ta55	Ta56	Ta57	Ta58	Ta59	Ta60
LB_1	3.34	1.77	0.99	3.92	3.83	4.90	2.36	9.53	2.37	2.87	2.91	1.87	2.36	2.53	2.95	2.92	2.97	2.93	0.94
LB_2 *AP*	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
LB_2 *Adj*	4.30	2.24	1.43	3.76	4.95	4.90	2.48	9.53	2.56	2.87	2.91	2.91	2.99	2.52	2.95	2.92	2.97	2.92	2.71
LB_2 *Last*	0.70	0.61	0.71	0.55	0.48	0.59	0.61	0.57	0.19	0.19	0.20	0.21	0.22	2.52	0.58	0.28	0.26	0.28	0.28
LB_2 *AT* ($k = m/2 + 1$)	0.43	0.65	0.60	0.55	0.48	0.40	2.48	1.36	0.17	0.14	0.13	0.14	0.15	0.15	0.14	0.17	0.13	0.12	0.19
LB_2 *AT* ($k = m$)	0.42	0.57	0.56	0.48	0.47	0.40	0.58	0.54	0.17	0.16	0.15	0.15	0.16	0.17	0.14	0.15	0.13	0.13	0.19
LB_2 *TA* ($k = m/2 + 1$)	0.76	0.95	0.65	0.50	1.39	0.95	2.48	0.56	0.49	0.19	0.14	0.43	2.99	2.52	0.70	2.92	0.19	0.13	0.41
LB_2 *TA* ($k = m$)	0.45	0.58	0.58	0.48	0.45	0.42	0.57	0.55	0.17	0.17	0.15	0.16	0.17	0.17	0.15	0.19	0.14	0.14	0.30
LB_2 *Ext-TA* ($k = m/2 + 1$)	0.80	62.92	0.65	13.24	1.69	4.90	2.48	1.37	1.36	2.87	0.14	2.91	2.99	2.52	2.95	2.92	2.97	0.17	0.18
LB_2 *Ext-TA* ($k = m$)	0.62	0.53	0.62	0.50	0.45	0.50	0.55	0.55	0.16	0.15	0.13	0.14	0.16	2.52	0.42	0.19	0.17	0.18	0.19
LB_2 *SB-TA* ($k = m/2 + 1$)	0.51	0.68	0.73	0.61	0.53	0.47	1.01	0.64	0.24	0.17	0.16	0.20	0.20	2.52	0.42	0.19	0.18	0.23	0.29
LB_2 *SB-TA* ($k = m$)	0.48	0.61	0.68	0.59	0.51	0.47	0.81	0.55	0.23	0.19	0.17	0.20	0.20	0.21	0.29	0.20	0.18	0.23	0.28
LB_2 *RS* ($k = 1$)	307.69	110.88	22.68	274.81	84.44	4.90	2.48	9.53	2.56	2.87	2.91	2.91	2.99	2.52	2.95	2.92	2.97	2.92	2.97
LB_2 *RS* ($k = m/2 + 1$)	2.25	39.02	9.62	109.38	84.44	4.73	2.48	7.40	1.99	1.97	2.91	2.91	2.23	2.52	2.95	2.92	2.61	2.92	2.60
LB_2 *RS* ($k = m$)	103.16	1.37	0.88	1.49	1.54	3.12	1.90	7.97	1.74	2.87	2.91	2.22	2.22	2.52	1.67	1.11	1.10	1.57	1.00
LB_2 *RR* ($k = 1$)	45.61	56.21	22.68	58.14	41.97	4.90	2.48	9.53	2.56	2.74	2.91	2.91	2.99	2.52	2.95	2.92	2.97	2.92	2.97
LB_2 *RR* ($k = m/2 + 1$)	2.07	2.25	2.15	2.58	2.56	3.01	2.48	4.59	1.85	1.42	2.59	1.88	2.62	2.52	2.76	2.20	2.97	2.64	1.76
LB_2 *RR* ($k = m$)	1.27	1.34	1.19	1.51	1.53	1.68	1.77	2.43	1.16	0.92	2.05	1.50	2.99	2.52	1.67	1.11	2.49	1.57	1.00
LB_2 *EWMA* ($k = m/2 + 1,\ \alpha = 0.05$)	0.44	0.66	0.61	0.56	0.49	0.41	2.48	1.38	0.17	0.14	0.14	0.15	0.16	0.16	0.15	0.18	0.13	0.13	0.19
LB_2 *EWMA* ($k = m/2 + 1,\ \alpha = 0.1$)	0.44	0.66	0.61	0.57	0.49	0.41	2.48	1.40	0.17	0.14	0.14	0.15	0.16	0.16	0.15	0.18	0.13	0.13	0.20
LB_2 *EWMA* ($k = m,\ \alpha = 0.05$)	0.43	0.58	0.58	0.49	0.47	0.40	0.59	0.55	0.17	0.17	0.15	0.16	0.16	0.17	0.15	0.16	0.14	0.14	0.19
LB_2 *EWMA* ($k = m,\ \alpha = 0.1$)	0.43	0.59	0.58	0.49	0.48	0.41	0.59	0.55	0.18	0.17	0.15	0.16	0.16	0.17	0.15	0.16	0.14	0.14	0.19
LB_2 *Champ* ($k = m/2 + 1$)	0.55	1.00	0.77	0.87	0.71	0.47	0.88	0.61	0.20	2.87	2.91	2.91	2.99	2.52	0.89	2.92	2.97	0.73	0.24
LB_2 *Champ* ($k = m$)	0.51	0.95	0.74	0.84	0.63	0.49	0.68	0.53	0.21	2.87	2.91	2.91	2.99	2.52	0.65	2.92	2.97	0.60	0.24
LB_2 *Shrink*	0.43	0.63	0.63	0.49	0.47	0.38	0.72	0.53	0.16	0.16	0.13	0.15	0.15	0.16	0.14	0.15	0.13	0.13	0.19
LB_2 *Knee*	0.42	0.62	0.60	0.56	0.47	0.39	0.72	0.85	0.17	0.14	0.13	0.14	0.16	0.16	0.15	0.15	0.13	0.12	0.18

Table A3 Standard deviation of runtime (seconds) on Taillard PFSP instances

Method	**Ta11**	**Ta12**	**Ta13**	**Ta14**	**Ta15**	**Ta16**	**Ta18**	**Ta20**	**Ta22**	**Ta51**	**Ta52**	**Ta53**	**Ta54**	**Ta55**	**Ta56**	**Ta57**	**Ta58**	**Ta59**	**Ta60**
LB_1	0.3	0.8	0.3	0.3	0.7	0.0	16.5	0.0	18.3	0.0	0.0	11.7	9.4	0.0	0.0	0.0	0.0	0.0	3.4
LB_2 *AP*	0.0	0.3	0.7	0.1	0.4	11.0	3.9	2.3	8.0	2.7	3.2	20.4	11.8	12.4	7.6	16.2	19.7	17.5	10.8
LB_2 *Adj*	0.3	1.0	1.8	0.5	2.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	55.0
LB_2 *Last*	0.2	0.1	1.2	0.1	0.4	1.9	3.1	2.2	0.9	1.1	1.7	4.0	1.9	0.0	3.6	6.5	1.7	1.0	5.5
LB_2 *AT* ($k = m/2 + 1$)	0.0	0.0	1.6	0.1	0.2	2.1	0.0	23.1	1.9	0.7	1.6	2.3	1.9	1.9	0.1	1.4	3.0	1.2	3.0
LB_2 *AT* ($k = m$)	0.1	0.1	0.5	0.1	0.3	5.5	5.6	8.2	1.3	0.6	1.9	2.8	0.3	3.7	2.0	0.3	2.2	2.6	2.3
LB_2 *TA* ($k = m/2 + 1$)	0.1	0.5	2.3	0.1	1.2	20.5	0.0	10.6	5.5	3.7	2.7	6.0	0.0	0.0	4.0	0.0	1.0	1.7	7.7
LB_2 *TA* ($k = m$)	0.0	0.3	0.5	0.0	0.2	6.3	7.0	9.7	1.4	1.2	1.6	1.1	0.7	1.8	1.9	1.0	1.7	0.8	3.8
LB_2 *Ext-TA* ($k = m/2 + 1$)	0.2	27	0.3	1.6	1.2	0.0	0.0	16.1	13.6	0.0	3.1	0.0	0.0	0.0	0.0	0.0	0.0	2.5	3.2
LB_2 *Ext-TA* ($k = m$)	0.1	0.2	1.1	0.1	0.2	6.4	9.0	10.5	2.8	1.0	1.0	2.6	1.1	0.0	3.0	1.1	2.8	2.4	2.6
LB_2 *SB-TA* ($k = m/2 + 1$)	0.2	0.5	1.4	0.1	0.7	12.5	3.2	12.9	2.6	2.0	2.5	3.1	1.4	0.0	0.3	1.2	2.7	3.9	5.9
LB_2 *SB-TA* ($k = m$)	0.0	0.3	1.3	0.2	0.4	8.5	23.3	11.7	1.1	2.0	3.7	2.5	1.6	3.2	3.1	1.5	3.0	4.5	4.1
LB_2 *RS* ($k = 1$)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
LB_2 *RS* ($k = m/2 + 1$)	35.5	6064.2	5474.7	3726.9	0.0	648.3	0.0	4184.8	4154.8	5867.1	0.0	0.0	4702.8	0.0	0.0	0.0	2254.7	0.0	1193.4
LB_2 *RS* ($k = m$)	6217.3	39.1	63.8	13.5	37.6	4872.3	2198.4	3072.4	3010.4	0.0	0.0	4430.0	4789.7	0.0	14.1	23.7	574.5	19.4	37.4
LB_2 *RR* ($k = 1$)	9.3	33.2	0.0	10.0	9.2	0.0	0.0	0.0	0.0	806.7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
LB_2 *RR* ($k = m/2 + 1$)	0.5	1.5	5.5	0.8	1.2	29.2	0.0	80.0	34.5	963.8	2044.8	19.7	2298.6	0.0	56.9	2313.4	0.0	39.8	17.9
LB_2 *RR* ($k = m$)	0.2	1.2	3.8	0.4	1.1	23.2	23.9	35.9	15.2	32.2	22.4	2060.8	0.0	0.0	14.1	23.7	2984.2	19.4	37.4
LB_2 *EWMA* ($k = m/2 + 1,\ \alpha = 0.05$)	0.0	0.0	1.4	0.0	0.5	2.5	0.0	3.4	1.4	0.9	0.4	1.0	1.9	2.4	1.3	1.2	1.6	0.7	1.4
LB_2 *EWMA* ($k = m/2 + 1,\ \alpha = 0.1$)	0.0	0.1	0.5	0.1	0.1	3.8	0.0	3.2	1.6	0.8	1.1	2.0	0.4	2.0	1.7	2.2	0.9	1.0	2.9
LB_2 *EWMA* ($k = m,\ \alpha = 0.05$)	0.2	0.1	5.9	0.1	0.1	4.1	8.0	1.4	0.8	0.7	0.5	2.9	0.7	2.1	1.4	0.8	1.9	1.0	2.1
LB_2 *EWMA* ($k = m,\ \alpha = 0.1$)	0.1	0.1	0.6	0.1	0.1	2.6	4.2	3.0	1.6	2.9	0.7	2.0	1.4	2.2	0.7	0.7	1.4	0.3	3.4
LB_2 *Champ* ($k = m/2 + 1$)	0.1	1.3	1.4	0.4	6.7	1.1	45.7	2.3	9.1	0.0	0.0	0.0	0.0	0.0	6.8	0.0	0.0	67.6	3.9
LB_2 *Champ* ($k = m$)	0.3	3.8	1.7	0.5	3.9	4.1	35.4	1.3	6.7	0.0	0.0	0.0	0.0	0.0	60.2	0.0	0.0	16.8	9.0
LB_2 *Shrink*	0.0	0.1	2.2	0.2	0.4	4.2	31.0	8.9	1.0	1.3	1.5	3.7	1.6	2.4	0.9	0.5	1.4	1.5	2.4
LB_2 *Knee*	0.0	0.2	2.0	0.2	0.3	3.0	31.0	15.7	1.6	1.7	2.2	3.3	1.9	2.7	0.5	1.1	1.6	1.4	0.5

References

Brown, A., Lomnicki, Z.: Some applications of the "branch-and-bound" algorithm to the machine scheduling problem. J. Oper. Res. Soc. **17**, 173–186 (1966) https://doi.org/10.1057/jors.1966.25

Brucker, P.: Scheduling Algorithms, p. 371. Springer, Heidelberg (2007)

Gmys, J., Mezmaz, M., Melab, N., Tuyttens, D.: A computationally efficient branch-and-bound algorithm for the permutation flow-shop scheduling problem. European Journal of Operational Research **284**(3), 814–833 (2020) https://doi.org/10.1016/j.ejor.2020.01.039

Helbecque, G., Gmys, J., Melab, N., Carneiro, T., Bouvry, P.: Parallel distributed productivity-aware tree-search using chapel. Concurrency and Computation: Practice and Experience **35**(27) (2023) https://doi.org/10.1002/cpe.7874

Helbecque, G., Gmys, J., Melab, N., Carneiro, T., Bouvry, P.: P3D-DFS: Parallel Distributed Depth-First Search Framework. GitHub repository (accessed 2026-04-20) (2024). https://github.com/Guillaume-Helbecque/P3D-DFS

Hunter, J.S.: The exponentially weighted moving average. Journal of Quality Technology **18**(4), 203–210 (1986)

Johnson, S.M.: Optimal two- and three-stage production schedules with setup times included. Naval Research Logistics Quarterly **1**(1), 61–68 (1954) https://doi.org/10.1002/nav.3800010110

Land, A.H., Doig, A.G.: An automatic method of solving discrete programming problems. Econometrica **28**(3), 497–520 (1960). Accessed 2026-04-17

Ladhari, T., Haouari, M.: Haouari, m.: A computational study of the permutationflow shop problem based on a tight lower bound. computers & operations research 32, 1831-1847. Computers & OR **32**, 1831–1847 (2005) https://doi.org/10.1016/j.cor.2003.12.001

Lageweg, B.J., Lenstra, J.K., Kan, A.H.G.R.: A general bounding scheme for the permutation flow-shop problem. Operations Research **26**(1), 53–67 (1978). Accessed 2026-04-17

Melab, N., Chakroun, I., Mohand, M., Tuyttens, D.: A gpu-accelerated branch-and-bound algorithm for the flow-shop scheduling problem. Proceedings - 2012 IEEE International Conference on Cluster Computing, CLUSTER 2012 (2012) https://doi.org/10.1109/CLUSTER.2012.18

Maudet, G., Danoy, G.: Search strategy generation for branch and bound using genetic programming. Proceedings of the AAAI Conference on Artificial Intelligence **39**(11), 11299–11308 (2025) https://doi.org/10.1609/aaai.v39i11.33229

Morrison, D.R., Jacobson, S.H., Sauppe, J.J., Sewell, E.C.: Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning. Discrete Optimization **19**, 79–102 (2016) https://doi.org/10.1016/j.disopt.2016.01.005

Mezmaz, M., Melab, N., Tuyttens, D.: 3. A Multithreaded Branch-and-Bound Algorithm for Solving The Flow-Shop Problem on A Multicore Environment, pp. 53–70. John Wiley & Sons, Ltd, Hoboken, NJ (2013). https://doi.org/10.1002/9781118640708.ch03

Nabeshima, I.: On the bound of makespans and its application in m-machine scheduling problem. Journal of the Operations Research Society of Japan **9**(3–4), 98–135 (1967)

Ruiz, R., Maroto, C.: A comprehensive review and evaluation of permutation flowshop heuristics to minimize flowtime. European Journal of Operational Research **40**, 479–494 (2005) https://doi.org/10.1016/j.ejor.2004.04.017

Taillard, E.D.: Benchmarks for basic scheduling problems. European Journal of Operational Research **64**(2), 278–285 (1993) https://doi.org/10.1016/0377-2217(93)90182-M

Talbi, E.-G.: Metaheuristics: From Design to

Implementation. John Wiley & Sons, Hoboken, NJ (2009). https://doi.org/10.1002/9780470496916

Tomazella, C.P., Nagano, M.S.: A comprehensive review of branch-and-bound algorithms: Guidelines and directions for further research on the flowshop scheduling problem. Expert Systems with Applications **158** (2020) https://doi.org/10.1016/j.eswa.2020.113556

Varrette, S., Bouvry, P., Cartiaux, H., Georgatos, F.: Management of an academic hpc cluster: The ul experience. 2014 International Conference on High Performance Computing & Simulation (HPCS), 959–967 (2014)

Vorokhta, A., Maudet, G., Danoy, G.: Selecting the best lower-bound strategy in a branch-and-bound algorithm using genetic programming. In: Blum, C., Cavallaro, C., Crespi, C., Talbi, E.-G. (eds.) International Workshop on Big Optimization 2025, pp. 48–63. Springer, Cham (2026)