RAIRO-Oper. Res. 60 (2026) 1981–2019
https://doi.org/10.1051/ro/2026049

RAIRO Operations Research
www.rairo-ro.org

EFFECT OF SKEWING STRATEGIES ON SOLVING SCHEDULING PROBLEMS FOR MULTIPROCESSOR TASKS ON TWO DEDICATED PROCESSORS

Fatma-Zohra Baatout[1], Naby Doumbouya[2] and Mhand Hifi[2,*]

Abstract. Efficiently scheduling multiprocessor tasks on two dedicated processors is a complex challenge. This is particularly true when navigating task dependencies and resource constraints. This paper investigates the impact of a skewed strategy within population-based metaheuristics. This strategy introduces a controlled tolerance for accepting degraded solutions. As a result, the exploration of the solution space is significantly enhanced. The proposed approach hybridizes a modified Grey Wolf Optimizer (GWO) with a skewed local search strategy. This combination takes advantage of the robust global search capabilities of the GWO while utilizing the skewed local search to maintain diversity. Consequently, the algorithm avoids premature convergence, which is a common issue in purely stochastic methods. A customized constructive procedure initializes the population. Solutions are then refined using specialized operators, such as a shaking mechanism and a "fix-and-complete" procedure. These components create a balance between exploration and exploitation. This balance allows the search to target promising regions while escaping local traps. Furthermore, the skewed acceptance criterion enables the exploration of slightly inferior paths. Such divergence from current trajectories reduces the risk of stagnation. To ensure stability across various problem instances, the algorithm's parameters were calibrated using the Taguchi method. This design of experiments framework systematically optimized key factors, including population size, iteration limits, and skewness tolerance. Experimental results on benchmark instances show that the proposed method outperforms existing state-of-the-art approaches. The achieved results were confirmed through rigorous statistical validation. The analysis included rank tests, the Wilcoxon signed-rank test, and analysis of variance.

Mathematics Subject Classification. 90B35, 90C59, 93A30.

Received December 21, 2024. Accepted May 8, 2026.

1. Introduction

Effective scheduling is a cornerstone of both engineering practice and academic research. Its importance stems from its direct impact on operational efficiency and resource utilization within complex systems. In practical scenarios, the application of advanced optimization techniques can significantly refine scheduling processes. This refinement ensures that tasks are executed with the highest possible degree of efficiency, directly influencing the productivity of industrial environments.

Keywords. Scheduling, knapsack, local search, skew.

[1] USTHB, LaROMaD, BP 32 El Alia, 16111 Alger, Algérie.
[2] UPJV, EPROAD UR 4669, 7 rue du Moulin Neuf, 80000 Amiens, France.
*Corresponding author: hifi@u-picardie.fr (All authors are listed in alphabetical order).

Beyond mere task execution, scheduling problems are ubiquitous across multiple domains. They remain integral to engineering, particularly regarding the optimization of resource allocation [30]. These challenges are prevalent in diverse industries, including manufacturing [19], transportation [28], and healthcare [34]. They also play a central role in telecommunications [33] and project management [16]. In these sectors, the primary objective is to optimize the use of limited resources, such as machines or processors. This must be achieved while strictly adhering to deadlines, priorities, and task dependencies. However, as the number of tasks and constraints grows, finding optimal solutions becomes increasingly difficult. This complexity often requires the use of sophisticated algorithms to handle the vast search spaces involved.

Building upon these industrial requirements, the present study focuses on a specialized class of problems known as Scheduling Tasks on Two Dedicated Processors (ST2P). This problem, previously investigated in [24] and [20], involves the assignment of n tasks to two processors, P_1 and P_2. While some tasks require only a single processor, others necessitate simultaneous execution on both, denoted as P_{12}. The objective is to develop a methodology to minimize the makespan ($C_{\max}$). This requires accounting for specific release times (r_j) and processing times (p_j) for each task $j \in \{1, \ldots, n\}$. Following the standard notation introduced by [15], the problem is formally represented as $P2|fix_j, r_j|C_{\max}$. Since this class of problems is classified as NP-hard in the strong sense, exact methods often fail to provide solutions within reasonable time frames for large-scale instances. This limitation necessitates the development of robust metaheuristics capable of finding near-optimal solutions efficiently.

By integrating these optimization methods into realistic operational frameworks, this research provides a systematic approach to the ST2P problem. Efficient task allocation does more than maximize throughput; it also reduces idle time and resource wastage. Ultimately, the synergy between engineering, optimization, and scheduling highlights the critical role of resource management. This connection is essential for achieving operational excellence and maintaining a competitive advantage across various industrial sectors.

1.1. Problem formulation with illustrative example

1.1.1. Problem formulation

The ST2P consists of scheduling a set of tasks across two processors, where:

- Some tasks must run exclusively on processor 1 (denoted P_1),
- Others run exclusively on processor 2 (denoted P_2),
- Some tasks require both processors P_1 and P_2 to be active simultaneously during their execution (denoted P_{12} or $P_{1\&2}$).

Each task j has a processing time p_j and a release time r_j, representing the earliest time at which it can begin. The completion time of task j is C_j, and the overall goal is to minimize the makespan $C_{\max}$ – the *"time at which all tasks are completed"*. The formal description of the model can be provided as follows:

$$\min \quad C_{\max} \tag{1}$$

$$C_j \geq C_i + p_j + (x_{ij} - 1)M, \quad \forall (i,j) \in (P_1 \cup P_{12}) \times (P_1 \cup P_{12}) \tag{2}$$

$$C_j \geq C_i + p_j + (x_{ij} - 1)M, \quad \forall (i,j) \in (P_2 \cup P_{12}) \times (P_2 \cup P_{12}) \tag{3}$$

$$x_{ij} + x_{ji} = 1, \quad \forall (i,j) \in (P_1 \cup P_{12}) \times (P_1 \cup P_{12}) \tag{4}$$

$$x_{ij} + x_{ji} = 1, \quad \forall (i,j) \in (P_2 \cup P_{12}) \times (P_2 \cup P_{12}) \tag{5}$$

$$C_{\max} \geq C_i, \quad \forall i \in P_1 \cup P_2 \cup P_{12} \tag{6}$$

$$C_i \geq r_i + p_i, \quad \forall i \in P_1 \cup P_2 \cup P_{12} \tag{7}$$

$$x_{ij} \in \{0,1\}, \quad \forall (i,j) \in (P_1 \cup P_{12})^2 \cup (P_2 \cup P_{12})^2 \tag{8}$$

where $x_{ij} = 1$ (decision variable) if task j starts after task i, and 0 otherwise. Constraints (2) and (3) enforce precedence between tasks on the same processor or both processors. Constraints (4) and (5) ensure that for

TABLE 1. An instance of 2STS: processor type, release time, and processing time.

	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}
Processor Type	P_1	P_2	P_{12}	P_1	P_2	P_{12}	P_1	P_2	P_1	P_2
Release Time r_j	0	0	1	0	0	2	0	1	3	2
Processing Time p_j	3	2	3	2	2	2	2	3	1	1

each task pair (i, j), either i precedes j or vice versa (no overlaps). Finally, constraints (6) and (7) ensure the makespan captures all task completions and that tasks respect release times.

From a computational complexity perspective, the scheduling problem $P_2|\text{fix}_j, r_j|C_{\max}$ addressed in this work is NP-hard in the strong sense [24]. The fundamental hardness of multiprocessor task scheduling on dedicated processors was initially established in [22], who showed that even the basic version without release dates is NP-hard for instances where more than five tasks require both processors simultaneously.

This computational intractability was further characterized in [18], where a definitive complexity mapping for multiprocessor tasks with pre-specified processor allocations were provided. The authors illustrated a sharp boundary in problem hardness: while minimizing the schedule length ($C_{\max}$) or the sum of completion times ($\sum C_j$) on two dedicated processors is polynomially solvable under basic conditions, the introduction of either precedence constraints or arbitrary release dates (r_j) – both of which are central to our studied model – renders the problem NP-hard in the strong sense.

Specifically, the simultaneous consideration of bi-processor tasks and arbitrary arrival times, as modeled in constraints (2)–(7), makes the non-preemptive case computationally intractable. As further highlighted in [24], exact solution methods for this problem configuration are generally limited to small-scale instances ($n \leq 30$), as the disjunctive nature of the processor allocation creates an exponentially growing search space. This motivates the development of the hybrid approach proposed in this study to handle larger, industrial-sized ST2P instances efficiently.

1.1.2. Illustrative example

Table 1 presents an instance composed of 10 tasks, each characterized by a release time, processing time, and processor requirement. Tasks are distributed across the two dedicated processors (P_1 and P_2) and a shared category (P_{12}) requiring simultaneous execution on both.

The initial scheduling sequence is established via a greedy positioning procedure based on the $r_j + p_j$ criterion. As illustrated in the Gantt chart (Fig. 1), where $C_{\max} = 15$, task start times are determined through a simulation that respects both processor availability and individual release constraints r_j. Multiprocessor tasks (*e.g.*, T_3 and T_6) requiring P_{12} act as synchronization barriers, being scheduled only when both processors are simultaneously idle. Conversely, processor-specific tasks are scheduled independently according to their respective resource timelines.

1.2. Contribution

This work presents a novel method for solving the ST2P with various operators and an innovative *skewing strategy*. The main objective of this hybrid approach, referred to as the *Skewed GWO*, is to enhance the performance of the standard GWO in addressing complex combinatorial problems, particularly multiprocessor task scheduling.

The proposed method aims to: (i) achieve a better balance between *diversification* (global exploration) and *intensification* (local exploitation), (ii) prevent *premature convergence*, which often leads to suboptimal solutions, and (iii) allow the controlled acceptance of slightly inferior solutions to escape local optima. Unlike classical acceptance mechanisms such as simulated annealing or threshold accepting, this controlled acceptance is driven by a distance-aware criterion based on the *normalized symmetric difference* between task schedules.

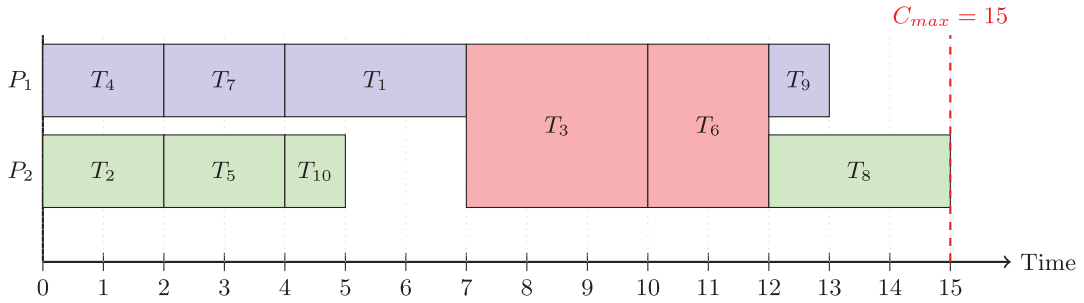

FIGURE 1. Gantt chart representing the specific task sequence for the illustrative example.

This formulation explicitly accounts for the *structural dissimilarity* between solutions, allowing inferior solutions to be accepted only when they are sufficiently different from the current one.

This strategy proves effective because, although the standard GWO shows strong exploratory behavior, it is prone to premature convergence in complex or multimodal landscapes. The integration of a *skewed local search* introduces a controlled form of diversification that:

- Encourages exploration of less promising regions that may conceal superior global solutions,
- Simulates adaptive behavior, enhancing the algorithm's ability to escape local optima,
- Improves overall robustness, particularly on large-scale or tightly constrained problem instances.

By combining the global search capabilities of the modified GWO with a skewed, adaptive local search mechanism, the approach achieves a robust balance between intensification and diversification. As a result, it consistently yields higher-quality solutions with improved computational efficiency, representing a significant advancement in scheduling optimization.

The proposed method consists of the following main components:

- *Greedy tailored initialization*: The algorithm begins with a tailored greedy initialization to generate an initial feasible task sequence. This foundational step strategically positions the search process within a promising region of the solution space.
- *Tailored sigmoid function*: A customized sigmoid function ensures the feasibility of solutions, particularly for tasks requiring allocation across both processors. This mechanism strengthens the algorithm's capabilities in navigating the feasible region efficiently, supporting effective convergence.
- *Variable Neighborhood Search (VNS)*: By combining a shaking-based 2-opt operator with a fix-and-complete procedure, the VNS intensifies exploration in promising regions and extends exploitation capabilities. This localized search refines task sequences, improving solution quality and uncovering near-optimal configurations.
- *Skewed strategy*: The method incorporates a new skewed operator as a tolerance mechanism. This strategy intentionally slows convergence to encourage exploration of neglected regions in the solution space. By accepting slightly inferior solutions that differ significantly from the current best according to a normalized structural distance, the skewed acceptance criterion jointly balances objective quality and solution diversity, facilitating the discovery of high-quality solutions that might otherwise remain undiscovered.

Extensive evaluations on benchmark instances show that the proposed approach significantly outperforms state-of-the-art algorithms for large-scale ST2P instances. It achieves superior solution quality while maintaining computational efficiency, solidifying its role as an innovative and robust contribution to scheduling optimization.

2. Related works

Scheduling remains a persistent challenge in combinatorial optimization, with applications spanning industries such as manufacturing, transportation, healthcare, and telecommunications. These problems capture a wide range of real-world scenarios and accurately reflect complex operational constraints, serving as a crucial basis for practical applications in academic studies. However, many scheduling problems are NP-hard, making the search for optimal solutions – whether for a single objective function or multiple Pareto-optimal points – particularly difficult. In this context, in [13], the author outlined the main variants of scheduling tasks on both parallel and dedicated multiprocessors, highlighting their complexity.

Due to the NP-hard nature of most scheduling issues, exact methods may only be feasible for small instances. As a result, the development of efficient (meta)heuristics is critical. For instance, an evolutionary method discussed in [30] highlights several strategies integrated into multi-population meta-heuristics. Moreover, this study also presents an enhanced version of these strategies tailored for efficient job scheduling in multiprocessor settings. Further extending this line of research, a thorough review of production scheduling solutions based on multi-population meta-heuristics has been provided in [23], establishing a classification to link various approaches.

Additional research has addressed specific variations of scheduling problems. In [8], the authors examined the case of scheduling tasks on two dedicated processors under preemptive constraints, denoted as $P2|\text{fix}_j, r_j, \text{pmtn}|C_{\max}$. An optimal solution procedure with polynomial time complexity was proposed for this setting. This study showed that although multiprocessor task scheduling is NP-hard in general, the introduction of preemption fundamentally changes the problem structure. In particular, allowing task interruption and resumption enables the derivation of low-order polynomial-time algorithms. The proposed approach relies on a linear programming formulation that efficiently integrates release times, processor availability, and preemptive execution, thereby identifying a polynomially solvable boundary within the broader class of multiprocessor scheduling problems.

In a subsequent study [10], a special case involving multiprocessor tasks on two identical parallel processors was analyzed. The authors investigated multiple restricted scenarios, including tasks with unit execution times, preemptive execution with release times and due dates, as well as precedence constraints. The objective functions considered included minimizing both the makespan and the maximum lateness, with detailed complexity results provided for each case. The study established a comprehensive complexity classification by identifying which combinations of task characteristics and constraints admit polynomial-time solutions and which lead to NP-hardness. In particular, it was shown that tractability can be preserved when either processing times are unitary or preemption is allowed, whereas the simultaneous presence of general processing times, release dates, and non-preemptive execution leads to computationally intractable variants. Beyond individual results, this work represents one of the earliest theoretical frameworks for systematically delineating the boundary between polynomially solvable and NP-hard subclasses in multiprocessor task scheduling on two processors.

Building on established tabu search methodologies, a sophisticated tabu search-based algorithm was designed in [32] to approximate solutions for multiprocessor scheduling problems. The approach integrates a tabu mechanism with a local search operator and effectively manages multiple solution lists throughout the search process. Key components of the algorithm include randomized blocking based on the tabu list size, frequency-based penalties to promote search diversification, and hashing techniques to retain high-quality solutions. Experimental results showed that specific combinations of these strategies led to substantial performance improvements, consistently outperforming existing algorithms on benchmark instances. The study further showed that random blocking of the tabu list tail plays a critical role in performance improvement, while frequency-based penalties are less effective than simple task-based tabu restrictions.

From a theoretical perspective, the work presented in [9] investigated multiprocessor task scheduling on three dedicated processors, providing a rigorous complexity analysis. The authors successfully proposed polynomial-time solutions for specific cases, such as those involving only uni-processor and duo-processor tasks. The strength of this work lies in its mathematical classification, which identifies tractable configurations solvable via linear

programming. However, the study lacks an experimental component, as the algorithms were not tested against practical benchmarks. Furthermore, these theoretical results do not scale to larger systems ($m > 3$) where the exponential increase in processor combinations necessitates the metaheuristic approaches explored in our current research.

A related contribution was provided in [11], where the authors developed two tabu search-based algorithms for multiprocessor scheduling on m dedicated processors. The strength of this work lies in its integration of local search with a tabu mechanism, which significantly improves solution quality over classical heuristics by effectively avoiding cycles during the intensification phase. However, the method's effectiveness is limited by its high sensitivity to the initial schedule; a weak starting point can lead to slow convergence or suboptimal local traps.

Moreover, the single batch-processing machine scheduling problem was addressed in [35] by taking into account variable job sizes and arbitrary arrival times. The authors designed a hybrid approach that integrated local search with path-relinking, utilizing three distinct types of local searches to improve solution diversity. The results demonstrated that path-relinking plays a key role in balancing intensification and diversification, allowing the method to outperform genetic algorithms and ant colony optimization on large-scale instances.

In the domain of scheduling tasks on two dedicated processors, in [24] an exact algorithm based on classical branch-and-bound techniques was designed. The algorithm incorporates specialized lower and upper bounds at each node of the search tree, demonstrating the ability to solve instances with up to thirty tasks in under fifty minutes. The study also showed that processor-relaxation-based lower bounds are tight for the preemptive case, enabling effective pruning through dominance rules.

Subsequently, a robust genetic algorithm was developed in [20] to approximate solutions for this problem. Instead of relying solely on standard evolutionary mechanisms, the authors combined traditional genetic operators with a constructive initialization, ensuring solution feasibility from the start. The algorithm employs a permutation-based encoding alongside rank-based selection, one-point crossover, and swap mutation to guide the search. To enhance quality and population diversity, the initial population is partially generated by a heuristic prioritizing tasks by release dates, with the remainder created randomly. Performance was evaluated using the framework established in [24]. The experimental study considers various task-type distributions and release-date tightness, providing a detailed assessment of robustness. Results indicate that the algorithm consistently produces near-optimal solutions and outperforms branch-and-bound as instance sizes grow.

In [4], a reactive search-based method was designed to enhance the scheduling process through adaptive mechanisms. The core strength of this work lies in the integration of a reactive tabu list that dynamically adjusts its tenure to avoid cycles, coupled with intensified local search operators such as 2-opt and 3-opt. Their computational study demonstrated that this reactive approach frequently improves upon the best-known bounds in the literature, particularly for large-scale instances. Nevertheless, the performance of the reactive search is heavily dependent on the fine-tuning of the tabu parameters; improper balancing between intensification and diversification can lead to premature convergence in complex search spaces.

In [5], the authors developed a hybrid method for the ST2P problem using a look-ahead strategy combined with path-relinking. The strength of this approach resides in its "intelligent" construction phase, which employs a knapsack-based greedy rule to prioritize tasks based on their long-term impact on the objective function, followed by a drop-and-rebuild diversification to escape local optima. Experimental results on 1440 benchmark instances confirmed its robustness, as the method reached optimal solutions for all small and medium cases. However, its effectiveness remains highly sensitive to the look-ahead depth: a shallow depth limits the foresight of the search, while an excessive depth significantly increases the computational overhead, making it less efficient for real-time scheduling scenarios.

Other studies have investigated specialized scheduling scenarios in order to bridge the gap between theoretical developments and industrial applications.

Specifically, mission scheduling in relay satellite systems was addressed in [31], where the authors proposed a knowledge-based evolutionary algorithm. Their approach is capable of handling complex resource constraints inherent to satellite operations. However, the overall performance of the method remains highly dependent

on the quality of the initial knowledge base, which may limit its adaptability in highly dynamic or uncertain environments.

Similarly, the study presented in [12] investigated consecutive multiprocessor job scheduling problems arising in the semiconductor industry. Their contribution lies in the development of both integer programming and constraint programming formulations. While these exact methods provide high-quality and optimal solutions, they face significant scalability challenges as the number of machines and jobs increases.

Furthermore, in [21], the authors studied scheduling problems involving fixed processor sets derived from initial graph configurations. Their solution is based on a Path Relinking–based metaheuristic designed to efficiently explore the search space. Experimental results obtained on benchmark instances demonstrated consistent improvements in solution quality. Nevertheless, the efficiency of the method is strongly influenced by the diversity of the elite solution set, which may lead to increased computational time when the set becomes excessively large.

Moreover, the distributed assembly no-idle flow-shop problem was examined in [36], where a cooperative water wave optimization algorithm incorporating reinforcement learning was proposed. The integration of reinforcement learning contributes to a better balance between exploration and exploitation during the search process. Despite these advantages, the proposed algorithm remains computationally complex when compared to simpler metaheuristic approaches, particularly in the context of large-scale manufacturing systems.

Several recent works have also explored hybrid metaheuristic approaches to solve complex combinatorial optimization problems.

In [3], an imperialist competitive algorithm was proposed for fuzzy p-hub center problems. The incorporation of an edge recombination operator together with the Elzinga–Hearn algorithm enabled the method to achieve optimal solutions for all 81 small-scale test instances. However, the method remains sensitive to the initial country distribution, which increases the risk of premature convergence.

Similarly, a variable neighborhood search approach was applied to many-to-many hub location–routing problems in [2]. One of its main strengths is its scalability, as it successfully solved instances involving up to 1000 nodes, whereas competing approaches failed beyond 440 nodes. Nevertheless, the convergence behavior may deteriorate in high-dimensional search spaces if the shaking phase does not provide sufficient diversification.

Finally, a hybrid genetic algorithm–simulated annealing approach was developed in [1] for competitive hub location and pricing problems. The proposed hybridization improved solution quality by 10% and reduced runtime by 9% on the Australia Post dataset when compared to a standard genetic algorithm. Despite these performance gains, the computational effort required to determine an effective cooling schedule remains a technical limitation.

The proposed method builds on previous work [7] and extends the initial experimental results presented therein. By explicitly targeting structural diversity within the search process, the proposed approach aims to further enhance solution robustness and improve scalability when addressing large ST2P instances.

3. Fundamental concept of GWO

The Grey Wolf Optimizer (GWO) is a meta-heuristic algorithm [25] inspired by the cooperative hunting behavior and social hierarchy of grey wolves in the wild. It simulates the wolves' social structure and hunting strategies to generate potential solutions for complex optimization problems. As a population-based approach, GWO shares similarities with other well-known metaheuristics, such as memetic algorithms, swarm optimization, and ant colony optimization. However, its distinguishing feature is the successive scattered search, which relies on a subset of solutions to iteratively evolve the population. In this framework, modulating the diversity of solutions plays a main role in balancing intensification and diversification, enhancing the search for optimal solutions. Moreover, this selective guidance mechanism allows GWO to progressively concentrate the search effort on promising regions of the solution space while still maintaining sufficient exploration capabilities.

In nature, grey wolves form packs with a clear hierarchical structure, consisting of the α wolf (the leader), the β wolf (the assistant), the δ wolf (a secondary role), and the ω wolves (the lowest-ranking members). This

hierarchy is not static, but rather adapts dynamically according to the wolves' performance during hunting. While the idea of a subset approach could be considered, for simplicity, each hierarchical rank is represented by a single wolf.

In the context of GWO, this hierarchy is translated into the optimization process by designating the α wolf as the best solution found so far. The β and δ wolves explore new regions of the solution space, while the ω wolves adjust their positions based on the guidance of the higher-ranked wolves, working in coordination to improve the overall solution quality. Consequently, information sharing among the leading wolves plays a central role in driving convergence toward high-quality solutions.

Grey wolf hunting tactics, including tracking, encircling, and attacking prey, inspire the main phases of GWO. As outlined in [29], these hunting behaviors influence the formulation of specific equations within the GWO. During the encircling phase, wolves employ directions to encircle prey, updating their positions based on the prey's location. This phase can be interpreted as a collective exploitation process guided by the estimated position of the prey. This process involves coefficient vectors $\overrightarrow{A}$ and $\overrightarrow{C}$, calculated as:

$$\overrightarrow{A} = 2\overrightarrow{a} \cdot \overrightarrow{r_1} - \overrightarrow{a} \quad \text{and} \quad \overrightarrow{C} = 2\overrightarrow{r_2}. \tag{1}$$

Additionally, each wolf's position is adjusted according to the prey's location using:

$$\overrightarrow{D} = |\overrightarrow{C} \cdot \overrightarrow{S_p}(t) - \overrightarrow{S}(t)|. \tag{2}$$

The updated position for each wolf is computed as:

$$\overrightarrow{S}(t+1) = \overrightarrow{S_p} - \overrightarrow{A} \cdot \overrightarrow{D}. \tag{3}$$

Herein, $\overrightarrow{S}(t+1)$ denotes the next iteration's position, $\overrightarrow{S_p}$ represents the prey's position, and $\overrightarrow{A}$ and $\overrightarrow{C}$ are coefficient vectors from equation (1).

The coefficient vector $\overrightarrow{a}$ in equation (1) controls the convergence speed towards the prey. It starts at 2 and decreases to 0 during iterations, influencing convergence rate. This gradual decrease ensures a smooth transition from exploration to exploitation throughout the optimization process. Conversely, $\overrightarrow{r_1}$ and $\overrightarrow{r_2}$ introduce randomness into wolf movements, aiding in avoiding local optima.

Given the difficulty in determining prey position, GWO approximates it with the pack leader's best position. Each iteration simulates grey wolves' hunting based on the three best solutions, namely α, β, and δ. These three solutions jointly define the search direction, reducing the risk of premature convergence caused by reliance on a single leader. Other wolves adjust their positions based on the best solution using:

$$\overrightarrow{D_\lambda} = |\overrightarrow{C_\lambda} \cdot \overrightarrow{S_\lambda}(t) - \overrightarrow{S}(t)|. \tag{4}$$

The three best positions are updated using:

$$\overrightarrow{S_1} = \overrightarrow{S^{(\alpha)}} - \overrightarrow{A_1} \cdot \overrightarrow{D_\alpha}, \quad \overrightarrow{S_2} = \overrightarrow{S^{(\beta)}} - \overrightarrow{A_2} \cdot \overrightarrow{D_\beta}, \quad \text{and} \quad \overrightarrow{S_3} = \overrightarrow{S^{(\delta)}} - \overrightarrow{A_3} \cdot \overrightarrow{D_\delta}. \tag{5}$$

Finally, each wolf's new position in the next iteration is calculated as the average of the positions of the three best wolves, forming a scattered position that is likely to lead to a better solution for the problem:

$$\overrightarrow{S}(t+1) = \frac{\overrightarrow{S_1} + \overrightarrow{S_2} + \overrightarrow{S_3}}{3}. \tag{6}$$

For clarity, $S^{(i)}$ denotes the solution of the ith wolf, while $S_1^{(i)}$, $S_2^{(i)}$, and $S_3^{(i)}$ represent the solutions associated with the three best wolves.

In the studied problem, a solution consists of an ordered sequence, such that the construction and evaluation of the solution depend critically on that order. This characteristic makes the problem particularly challenging,

as small changes in ordering may lead to significant variations in solution quality. For example, in P_{ST2P}, each candidate solution represents a sequence of elements (*e.g.*, tasks, trips, or locations), and the quality of the solution is directly affected by the relative ordering of these elements. Thus, the optimization process must explore not only which elements to include, but also how to arrange them, making order-sensitive search mechanisms essential. Accordingly, the ability of GWO to balance structured guidance and solution diversity makes it well suited for tackling such permutation-based optimization problems.

4. A skewed population-based method

The primary aim of the population-based method is to tackle the P_{ST2P} by employing a population of solutions $\mathcal{P}$, which simulates the grey wolf population. Subsequently, employing a variety of strategies, the method aims to guide the search process and enhance the quality of solutions. Through the integration of these elements, the method aims to provide an efficient approach that is able to provide good approximate solutions for the P_{ST2P}. Moreover, the proposed population-based framework is designed to adaptively combine collective learning with controlled diversification, allowing the search process to better cope with the combinatorial nature of the problem.

Herein, the skewed local search complements the global exploration of GWO by accepting slightly inferior yet diverse solutions, thereby reducing the risk of premature convergence and enhancing diversification. In particular, this acceptance mechanism deliberately trades off solution quality against structural diversity, ensuring that promising alternative regions of the search space are not prematurely discarded. Such a procedure will be illustrated on example for a better comprehension (*cf.*, Sect. 4.5.4).

4.1. The generational method

The initial population $\mathcal{P}_0$, also known as the starting *reference set*, is of paramount importance in any population-based method as it serves as the foundation for subsequent improvements. It should consist of both high-quality solutions and a diverse range of solutions. High-quality solutions enable exploration within the local neighborhood, while diversity among solutions facilitates exploration across various solution subspaces, ultimately contributing to the construction of superior-quality solutions. Consequently, the overall effectiveness of the proposed method strongly depends on the careful balance between quality and diversity achieved at this initialization stage.

4.1.1. Generating the initial greedy solution for the reference set

The first candidate for the reference set $\mathcal{P}_0$ (initially $\mathcal{P}_0 = \{\}$) can be obtained using a two-stage Tailored Greedy Procedure (TGP) for scheduling. This procedure is intended to rapidly generate a feasible and competitive solution that serves as a strong anchor for the subsequent population construction. Such a procedure GP works as follows:

(1) *Task rearrangement:* Tasks are rearranged based on a specified criterion.
(2) *Step-by-step task assignment:* Tasks are systematically assigned to processors one by one.

In the second step above, all tasks are iteratively placed on their respective processors. The scheduling procedure relies on the following characteristics (where r_j denotes the release date of task j, and p_j denotes its processing time):

(1) *Ratios:* Compute the ratio of processing time to release date for each task, *i.e.*, $\frac{p_j}{r_j}$ for all $j \in N$.
(2) *Rank tasks:* Arrange all tasks in non-increasing order of their ratios, *i.e.*, $\frac{p_1}{r_1} \geq \frac{p_2}{r_2} \geq \cdots \geq \frac{p_n}{r_n}$.

This ranking criterion prioritizes tasks that combine high processing requirements with early availability, which is particularly effective for constructing compact schedules. By iteratively arranging tasks based on this order, an initial solution can be generated. Such a solution, included in $\mathcal{P}_0$, represents a sequence of tasks allocated to either the first processor, the second processor, or both processors. Despite its simplicity, this initial sequence captures essential structural properties of high-quality schedules.

4.1.2. Augmenting the reference set $\mathcal{P}_0$

The iterative process involves generating a new configuration by randomly swapping two positions within the sequence established by TGP (*cf.*, Sect. 4.1.1). The process is repeated, resulting in the construction of multiple sequences. Such random perturbations introduce controlled variability into the population, which helps prevent the concentration of solutions around a single construction pattern.

Let $X_\circ$ represent the initial solution obtained by applying TGP, with "pos_1" and "pos_2" denoting two randomly generated positions within the current selected sequence (initially, the sequence built from TGP is used). Then, a permutation is performed between these positions followed by the reapplication of TGP to generate a new solution. By reapplying the greedy assignment after each swap, feasibility is preserved while exploring alternative task orderings. The following steps outline the procedure for establishing a new sequence:

(1) Place all tasks designated for the first processor.
(2) Proceed by arranging the tasks designated for the second processor.
(3) Distribute the remaining tasks to both processors.

Let T denote the resulting sequence, and Score_{X_i} (resp. Score_{T_i}) represent the score associated with the position of the task in the current solution (sequence), defined as:

$$\text{Score}_{X_i} = m - i + 1 \tag{7}$$

where $i = 1, \ldots, m$. This scoring scheme assigns greater importance to tasks appearing earlier in the sequence, reflecting their stronger influence on the overall schedule quality. By considering the resulting orders of both X and T, the scores associated with both solutions can be summed to obtain a new solution, denoted as X', integrated to the current population $\mathcal{P}_0$. Such a new solution arranges the tasks in descending order of their scores. This aggregation mechanism effectively merges information from different solutions, producing hybrid sequences that inherit favorable characteristics from each parent.

Such a process can be iterated until either the final population ($\mathcal{P} := \mathcal{P}_0$) is generated or a subset of solutions is obtained with a size smaller than the desired population size. In practice, this allows the reference set to be progressively enriched with solutions that are both competitive and structurally diverse. Note that to create a complete initial population of solutions, a swapping operator is also introduced for the current generated solutions. Hence, the solution $X_\circ$ may be randomly generated from the current created reference set $\mathcal{P}$ and, at each generation whenever the provided solution is new, then $\mathcal{P}$ is updated with that solution. This continuous update mechanism further reinforces diversification and reduces the risk of early stagnation in the search process.

4.2. Determining new positions

Herein, the Positioning Procedure (PP) is used to determine the successive positions induced by individual wolves during the search process. PP involves combining the current wolf's position with the positions of the three best wolves α, β, and δ in the current population $\mathcal{P}$ of solutions, aiming to create a scattered configuration/solution. Moreover, this mechanism allows each wolf to benefit simultaneously from collective guidance and individual exploration, which is essential for navigating complex combinatorial landscapes.

Recall that equation (6) often yields a fractional value, which is used to compute the new scatted position of the wolf at hand. The new provided position depends on the position of the virtual prey, simulated by the combination of the three best positions, and the last position of the respective wolf. Consequently, the resulting position can be interpreted as a weighted compromise between past experience and current global guidance.

Herein, a series of fractional positions is considered, where each ith integral position (solution) of the population $\mathcal{P}$ is obtained using the following transformation based on the newly achieved position. This transformation serves as a bridge between the continuous search space induced by GWO and the discrete, permutation-based nature of the studied problem. Let X represent a wolf's position, an individual within the population, represented as a sequence of tasks (nonnegative integer numbers). The new position X' of the corresponding wolf

(with the actual position X) can be defined by applying the following function F, which is an adaptation of the sigmoid function:

$$F_i = \left(\frac{1}{1+e^{-X_i}}\right) \times r_i \tag{8}$$

where F_i represents the coefficient measuring the degree of importance of the ith task in the new achieved sequence, and r_i represents the arrival time of task i. The incorporation of arrival times biases the transformation toward tasks that are both structurally significant and temporally critical. By ordering F_i, $i = 1, \ldots, n$, in decreasing order, a new sequence X' (leading to a feasible solution) can be obtained, related to the new order. Hence, higher-ranked tasks are more likely to be scheduled earlier in the resulting sequence.

The process of transforming the current sequence X, which represents the position of a wolf, into a new position X' involves two steps. These steps ensure that feasibility is restored while preserving the exploratory information embedded in the fractional position. Then,

(1) Initially, the sequence X is subjected to the sigmoid function adaptation, using equation (8) to compute the values of the function F. These values are determined based on the task positions in sequence X and their associated arrival times.
(2) Subsequently, using the values of F, a new sequence X' is derived. Such a new sequence reflects a probable sequence for the same wolf.

Hence, the first component of X' corresponds to the task with the highest value of F, while the last position in the new sequence represents the task with the smallest value of F. This ordering mechanism naturally emphasizes tasks with greater influence on the overall scheduling performance.

4.3. Feasibility handling

It is worth noting that while GWO can operate with infeasible positions (fractional positions), the problem P_{ST2P} studied does not necessitate the feasibility of each agent's position. Instead, infeasible intermediate representations are deliberately exploited to enhance exploration. As discussed in Section 4.2, a score vector can be used to induce a sequence, from which a new assignment can be derived. Suppose $C^{(i)}$ represents the current vector position (a configuration) of a given agent i, obtained through the scattered solution:

$$C^{(i)} = \frac{X_1^{\alpha} + X_2^{\beta} + X_3^{\delta}}{3}, \tag{9}$$

where X_1^{α}, X_2^{β}, and X_3^{δ} denote the positions provided by applying equation (5). This averaged position encodes collective knowledge gathered from the leading wolves. Recall that the modified sigmoid function based on the scoring function equation (8) is applied to $C^{(i)}$ for establishing the new current position, denoted as $X^{(i)}$. Thus, feasibility is restored only after the collective information has been fully exploited. The process involves the following steps:

(1) Let $C^{(i)}$ be the current configuration related to agent i.
(2) Apply equation (8) to $C^{(i)}$, and let $F^{(i)}$ be the resulting vector.
(3) Reorder $F^{(i)}$ in decreasing order of their values and reorder the tasks related to $C^{(i)}$ accordingly.
(4) From $C^{(i)}$, apply the procedure described in Section 4.1.1 to build a new solution, denoted as $X^{(i)}$.

As a result, each agent alternates between infeasible exploratory moves and feasibility-restoring reconstruction steps, ensuring both diversity and solution validity throughout the search process.

4.4. Exploitation search

While the Positioning Procedure (PP) promotes diversification by generating substantially perturbed sequences, an explicit exploitation mechanism is required to counterbalance this effect and refine solution quality. To this end, an exploitation search is introduced to stabilize promising solutions, intensify the search around

high-quality regions, and guide the overall process toward convergence. In practice, this balance between diversification and exploitation is essential to avoid both premature convergence and excessive randomness during the search.

In this work, and in line with the strategies proposed in [5] and [6], the first exploitation operator relies on a simple yet effective *swapping* mechanism. This operator generates a neighboring solution by exchanging the positions of two tasks in the current sequence. Moreover, multiple swaps can be applied iteratively, enabling a progressive exploration of alternative solutions within the same neighborhood. As a result, this operator adjusts the neighborhood structure dynamically, allowing the search to fine-tune task ordering while preserving feasibility. Furthermore, the low computational cost of swap moves makes them particularly suitable for frequent use within an iterative improvement framework.

Furthermore, to further enhance solution quality, a second exploitation technique is employed based on partial assignment. In this approach, a subset of tasks is temporarily fixed to specific processors, producing a partial solution. The remaining tasks are then reassigned to complete the schedule. This strategy, referred to as the *fix-and-complete operator*, closely resembles the drop-and-rebuild mechanism described in [6]. By selectively fixing high-impact tasks while reorganizing the others, the operator enables deeper restructuring of solutions without losing valuable information. Consequently, this operator allows the search to escape shallow local optima that cannot be overcome through simple swap-based moves alone.

By combining the *swapping* and *fix-and-complete* operators, a descent-based exploitation strategy is constructed. This hybrid mechanism iteratively refines solutions by alternating between fine-grained local adjustments and more disruptive yet controlled reconstructions. Consequently, the search benefits from both precision and flexibility when navigating the solution space. As a result, the exploitation phase remains both robust and adaptable to different solution landscapes.

4.4.1. Variable neighborhood descent

To handle the exploitation phase, a Variable Neighborhood Descent (VND) scheme is employed as a core local improvement component of the proposed approach. Rather than detailing the full pseudocode in the main text, we provide here a concise description of the procedure, while the complete algorithmic formulation is reported in Section A.1 (Appendix A).

The VND procedure operates according to the following principles:

- It starts from an initial feasible solution X_0.
- A predefined list of neighborhood structures $\{N_1, \ldots, N_{k_{\max}}\}$ is explored sequentially.
- Within each neighborhood, a first-improvement local search is applied to identify an improving neighbor.
- When an improving solution is found, the current solution is updated and the search is restarted from the first neighborhood.
- If no improvement is obtained in the current neighborhood, the algorithm moves to the next one.
- The procedure terminates when all neighborhoods are explored without improvement, yielding a solution that is locally optimal with respect to the entire neighborhood set.

This restart mechanism allows the method to fully exploit each neighborhood structure while maintaining a low computational overhead. As highlighted in [27], the order in which neighborhoods are explored can significantly affect the efficiency of the VND. In this study, preliminary experiments (*cf.*, Sect. 5.2.3) showed that applying the *"swap"* neighborhood first, followed by the *"fix-and-complete"* neighborhood, leads to the most effective exploitation behavior. This ordering naturally progresses from small local modifications to more disruptive solution restructurings.

Note that the complete pseudocode of the basic VND procedure, referred to as Algorithm 2, is provided in Appendix A for completeness.

4.4.2. Swapping operator

Generally, the 2-opt operator repeatedly performs swaps (or "shakes") on a given feasible solution as long as the quality of the resulting solution improves. However, when the search process stagnates around the

same objective value, it often indicates that the 2-opt operator has reached its limit and is trapped in a local optimum. This situation typically arises when the local neighborhood induced by simple swaps no longer contains improving solutions.

Herein, a swapping operator based random 2-opt procedure is used that swaps two randomly chosen positions in the current sequence. Recall that the swaps induce a neighborhood around the current solution, where swapping two tasks may lead to either a feasible or an infeasible solution. Introducing randomness at this stage increases the likelihood of discovering alternative promising configurations. Because the proposed method considers only feasible solutions, then in the case of an infeasible solution, a greedy repairing operator is applied to make it feasible. The swapping operator consists of two steps:

(1) In the first step, the following actions are performed: (a) based on the position of task i, move all tasks to the right until all overlapping is resolved and, (b) based on the position of task j (after the swap), move all tasks to the right until all overlapping is resolved.
(2) In the second step, the greedy procedure (*cf.*, Sect. 4.1.1) is applied to the resulting order, as the swapping operator produces a new sequence.

By applying the above steps to the current solution, a series of solutions can be generated, forming the 2-opt neighborhood structures. Hence, the swapping operator acts as a lightweight yet effective local improvement mechanism.

4.4.3. Avoid premature convergence

Often, repeatedly applying 2-opt operators can lead to convergence towards the same local optimum, resulting in premature convergence. This phenomenon is particularly common in tightly constrained scheduling problems. To overcome this limitation, we employed a combination of both operators and a tabu list [14]. The tabu list acts as a temporary record of reverse swaps, preventing the algorithm from revisiting previously explored solutions. By forbidding recently executed moves, the search is explicitly encouraged to explore new regions of the solution space.

To control the growth of the tabu list and avoid excessive memory usage, a fixed-size tabu tenure strategy was implemented. Rather than storing entire solutions or objective values, the method maintains a dynamic list of recently applied moves, each assigned a predefined tenure. This tenure determines the number of iterations a move remains prohibited. Once the tenure expires, the move is removed from the list, allowing it to be reconsidered. This strategy ensures a balance between diversification and computational efficiency. In our computational experiments, a tenure of 10 iterations provided a good balance between intensification and diversification, improving convergence stability without significantly increasing computational overhead.

4.5. Exploration search

To escape local optima, the search process can be diversified through perturbations or "shaking" to explore different regions of the solution space. Such diversification mechanisms naturally complement exploitation-oriented operators. Methods such as larger neighborhood structures, random restarts, or hybrid approaches combining 2-opt with global strategies like VNS effectively overcome stagnation and enhance optimization performance.

This approach involves two types of operators. The first uses a random *fix and complete strategy*, where some tasks are assigned randomly, and the rest are completed using a tailored greedy procedure. The second relies on a *skew strategy*, which promotes asymmetric movements to explore less intuitive regions and enhance search diversification. Together, these operators introduce controlled disruptions that broaden the scope of the search.

4.5.1. Shaking procedure

First, a *"random fix-and-complete"* operator is introduced to extend exploration and reduce premature convergence. This operator deliberately perturbs the current solution while preserving part of its structure, thereby promoting controlled diversification.

The shaking procedure intentionally removes selected elements from the current solution in order to create room for exploration. The resulting partial solution is then reconstructed and optimized, allowing the search to move toward new regions of the solution space. This mechanism plays a central role in escaping local optima while maintaining solution feasibility.

Such a shaking procedure can be summarized as follows:

- *Partial destruction:* a given percentage β of (task, processor) pairs is randomly fixed, while the remaining part of the solution is removed.
- *Reconstruction:* the resulting partial solution is completed using an optimization procedure similar to the one employed for initial solution construction (*cf.*, Sect. 4.1.1).
- *Diversification effect:* the reconstructed solution differs structurally from the original one, enabling the exploration of new promising areas.

Note that the complete pseudocode of the shaking procedure, including a detailed line-by-line description, is provided in Section A.2 (Appendix A), where Algorithm 3 formally describes the fix-and-complete operator.

4.5.2. Employing a skewing strategy with normalized acceptance

Skewed VNS (SVNS) differs from traditional VNS [26] by allowing, under specific conditions, the acceptance of slightly worse solutions that are structurally dissimilar to the current one. Unlike classical VNS, which strictly favors improving moves, SVNS explicitly incorporates solution diversity into the acceptance decision. This relaxation of strict improvement criteria is crucial for escaping deep local optima and for sustaining progress in highly constrained or multimodal search spaces.

The acceptance rule employed in this work is defined as follows:

$$\frac{z(S)}{z(S')} + \rho \cdot d(S', S) > 1, \tag{10}$$

where $z(S)$ denotes the objective value of the current solution, $z(S')$ that of the candidate solution, and $d(S', S) = \frac{|S' \,\Delta\, S|}{|S' \cup S|}$ represents the normalized symmetric difference between the task schedules of S and S'. This formulation simultaneously accounts for solution quality and structural deviation.

Interpretation: The acceptance criterion establishes an explicit trade-off between intensification and diversification. The ratio $\frac{z(S)}{z(S')}$ measures the relative improvement (or degradation) in objective value, while the term $\rho \cdot d(S', S)$ introduces a controlled *"bonus for structural dissimilarity"*. As a result, a candidate solution that is slightly inferior in quality may still be accepted if it is sufficiently different from the current one, thereby encouraging exploration of new regions of the solution space.

Parameter Control: The parameter ρ regulates the sensitivity of the acceptance rule to structural changes. It therefore acts as a diversification control parameter, directly influencing the balance between accepting quality improvements and promoting structural diversity. Larger values of ρ favor broader exploration, whereas smaller values bias the search toward intensification around the current solution.

4.5.3. An adaptive skewed VND procedure

The adaptive skewed VND procedure combines local intensification and controlled diversification by integrating the shaking operator (Sect. 4.5.1) with the Skewed Variable Neighborhood Search (SVNS) mechanism (Sect. 4.5.2). This hybrid design enables the search to alternate dynamically between improving high-quality solutions and exploring structurally different regions of the solution space.

Starting from an initial solution X, the procedure iterates over a predefined number of global iterations G_{iter}. The best solution found so far, denoted by X_{best}, is initialized to X and updated throughout the search.

The main steps of the adaptive skewed VND procedure are summarized as follows:

- A local intensification phase is performed using a VND (*cf.*, Sect. 4.4.1), allowing the algorithm to efficiently exploit promising regions.
- Whenever an improving solution is identified, it is immediately accepted and the neighborhood exploration is restarted (*cf.*, Sect. 4.5.2), thereby reinforcing intensification.
- If no direct improvement is found, a skewed acceptance criterion is applied, which jointly evaluates solution quality and structural dissimilarity. This mechanism allows the acceptance of slightly inferior but sufficiently different solutions, thus promoting controlled diversification.
- A shaking phase based on the fix-and-complete operator is periodically applied (*cf.*, Sect. 4.5.1) to generate diversified starting solutions and facilitate escapes from local optima.
- The search process continues until the global iteration limit is reached, and the best solution encountered during the search is returned.

Note that the complete pseudocode of the adaptive skewed VND procedure, together with a detailed line-by-line explanation, is reported in Section A.3 (Appendix A).

4.5.4. Explanation and example of the adaptive skewed strategy

(A) Summary of the skewed method

Recall that the adaptive SVND enhances traditional VND by incorporating a normalized acceptance criterion. This mechanism allows the algorithm to occasionally accept worse solutions if they are *"structurally dissimilar"* to the current one. Unlike classical acceptance rules that rely solely on objective degradation (*e.g.*, simulated annealing or threshold accepting), the proposed mechanism explicitly integrates a measure of structural dissimilarity, thereby ensuring that inferior solutions are accepted only when they lead the search toward unexplored regions of the solution space. The method can be summarized as follows:

(1) Start from an initial solution S.
(2) Define a sequence of neighborhood structures $\mathcal{N}_1, \mathcal{N}_2, \ldots, \mathcal{N}_k$.
1. For each neighborhood $\mathcal{N}_i$:
   - Generate a candidate solution $S' \in \mathcal{N}_i(S)$.
   - If $f(S') < f(S)$: accept S' (as it's a better solution for a minimization problem).
   - Otherwise (if $f(S') \geq f(S)$, meaning S' is worse or equal):
     - Compute the distance $d(S', S)$, defined as the normalized symmetric difference.
     - Accept S' if the skewed condition (10) holds.

   where $\rho \in (0, 1]$ is a parameter controlling the *bonus awarded for structural dissimilarity.*
(3) Adapt ρ based on search progress (*e.g.*, increase if stagnation is observed to amplify the dissimilarity bonus and promote diversification).
(4) Repeat until a stopping criterion is met (*e.g.*, no further significant improvements are observed across a set number of iterations or neighborhoods, or a maximum execution time is reached).

From an optimization standpoint, this adaptive mechanism naturally alternates between intensification and diversification phases. When improvements are frequent, better solutions are systematically accepted, reinforcing intensification. Conversely, when stagnation occurs, the distance-based term becomes dominant, enabling exploratory moves toward structurally distant solutions while preventing uncontrolled deterioration of solution quality.

(B) Skewed strategy acceptance rule – example

The following example illustrates the application of the skewed acceptance rule (*cf.* Eq. 10), where the current solution is denoted by S, and two candidate solutions S' and S' are evaluated. This example aims to highlight how the skewed criterion balances objective quality and structural diversity, and how this balance influences acceptance decisions. The skewed rule relies on the following terms:

- $z(S)$: Makespan (*i.e.*, $C_{\max}$) of the current solution,

TABLE 2. Task characteristics for the 16-Task ST2P example.

	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}	T_{11}	T_{12}	T_{13}	T_{14}	T_{15}	T_{16}
Type	P_1	P_2	P_1	P_2	P_1	P_2	P_1	P_2	P_1	P_2	$P_{1,2}$	$P_{1,2}$	$P_{1,2}$	$P_{1,2}$	$P_{1,2}$	$P_{1,2}$
p_j	4	2	3	5	2	4	3	2	5	3	6	4	3	2	5	1
r_j	0	1	2	0	3	2	1	4	0	2	0	1	2	3	0	1

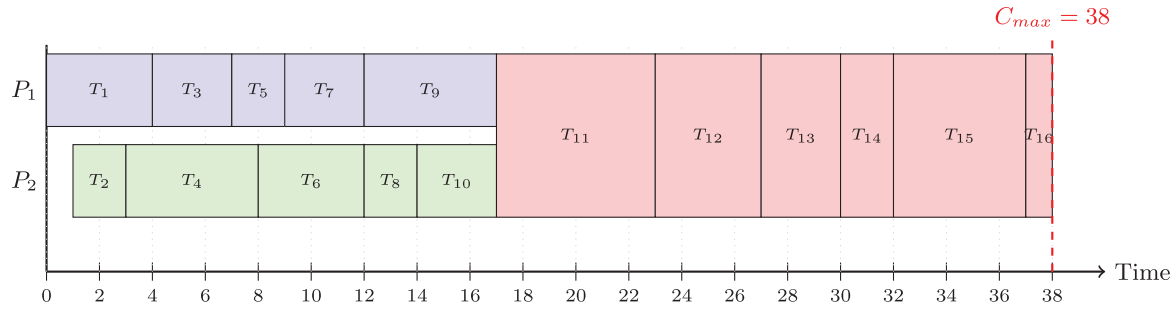

FIGURE 2. Gantt representation of the current solution S.

– $z(S')$: Makespan of the candidate solution,
– $d(S', S)$: Normalized Hamming distance between S' and S,
– ρ: Skewing tolerance parameter.

The normalized distance is scale-invariant, meaning that the acceptance behavior of the skewed rule remains consistent across different problem sizes. This property is particularly important for large-scal ST2P instances.

The effectiveness of the skewed strategy is illustrated through a comparative analysis of candidate solutions generated from a 16-task instance. The task characteristics for this instance are summarized in Table 2.

At a given iteration, the skewed strategy evaluates two candidates with respect to the current solution S. Unlike traditional local search methods that only accept improving moves, the skewed strategy employs a specialized acceptance criterion. This criterion balances solution quality (measured by $C_{\max}$) and structural diversity (measured by the normalized distance $d(S', S)$). A degraded solution may still be accepted if it is sufficiently different from the current state, thereby allowing the metaheuristic to escape local optima and explore distant, promising regions of the search space.

(1) Current solution S (*cf.*, Fig. 2):

`['T1', 'T2', 'T3', 'T4', 'T5', 'T6', 'T7', 'T8', 'T9', 'T10', 'T11', 'T12', 'T13', 'T14', 'T15', 'T16']`
$C_{\max}(S) = 38$, as illustrated in Figure 2. This solution represents a high-quality packing where dedicated tasks are efficiently overlapped, and multiprocessor tasks follow sequentially with minimal idle time.

(2) Candidate S'_1 (accepted despite higher cost – Fig. 3):

`['T2', 'T1', 'T4', 'T12', 'T3', 'T7', 'T8', 'T6', 'T10', 'T9', 'T11', 'T5', 'T14', 'T13', 'T16', 'T15']`
$C_{\max}(S'_1) = 42$ (Fig. 3), with a normalized distance: $d(S'_1, S) = \frac{15}{16} = 0.938$
Skew acceptance check for $\rho = 0.2$:

$$\frac{38}{42} + 0.2 \cdot 0.938 = 0.9048 + 0.1876 = 1.0924 > 1$$

→ Accepted.

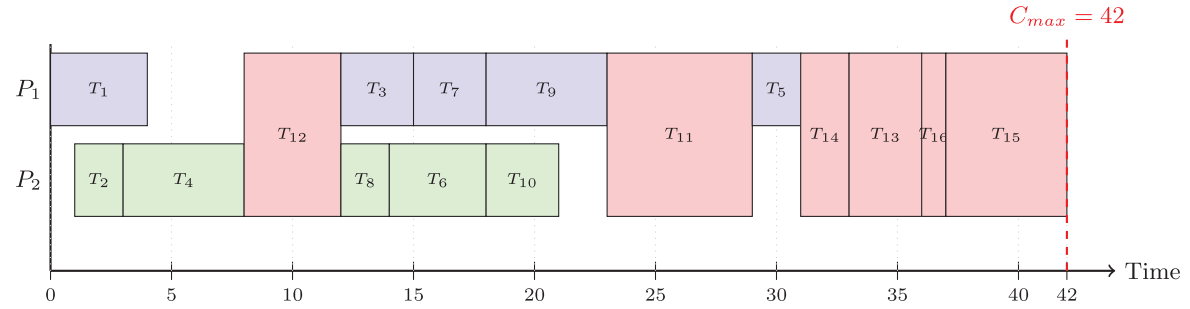

FIGURE 3. Gantt chart for candidate solution S_1' exhibiting a degraded makespan.

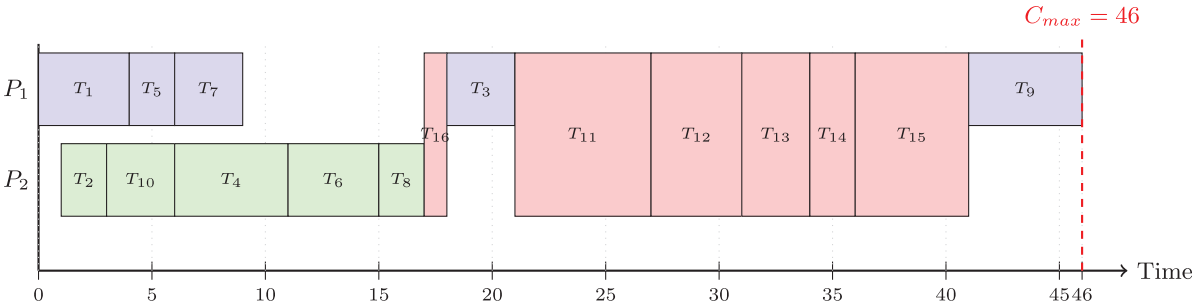

FIGURE 4. Gantt chart for candidate solution S_2', showing significant idle time and rejected makespan.

As depicted in Figure 3, S_1' presents a significant restructuring of the task sequence, particularly with the early insertion of the multiprocessor task T_{12}. Although this creates a makespan degradation of 4 units, the normalized distance is extremely high (0.938). This suggests that S_1' is structurally very different from S. The skewed strategy recognizes this as a valid diversification step, accepting the solution to explore a different topological region of the scheduling landscape.

(3) Candidate S_2' (rejected – Fig. 4):

`['T2', 'T1', 'T10', 'T4', 'T5', 'T6', 'T7', 'T8', 'T16', 'T3', 'T11', 'T12', 'T13', 'T14', 'T15', 'T9']`
$C_{\max}(S_2') = 46$ (Fig. 4)
Normalized distance: $d(S_2', S) = 0.375$
Skew acceptance check for $\rho = 0.2$:

$$\frac{38}{46} + 0.2 \cdot 0.375 = 0.826 + 0.075 = 0.901 < 1$$

→ Rejected.

This example highlights the selectivity of the skewed acceptance rule. Although candidate S_1' has a worse makespan, it is accepted due to its high structural dissimilarity, which steers the search toward a new region of the solution space. In contrast, candidate S_2' is both inferior in quality ($C_{\max} = 46$) and too structurally similar to the current solution ($d = 0.375$). As shown in Figure 4, the poor task sequencing leads to excessive idle time without offering enough diversification benefit. Consequently, it fails the skew acceptance check. This behavior proves that the skewed strategy promotes diversification without blindly accepting all degraded solutions; it only accepts those that offer high potential for exploration.

4.6. An overview of the skewed method

By incorporating the skewing strategy into GWO, this approach enhances both the exploration and exploitation capabilities of the algorithm, leading to better schedules for multiprocessor task scheduling problems. The skewed mechanism is mainly activated during stagnation phases. It preserves population diversity and enables controlled escapes from deep local optima. As a result, premature convergence is effectively avoided.

Algorithm 1. The skewed population-based method.

1: Generate an initial solution X using the greedy procedure (*cf.*, Sect. 4.1.1).
2: Build a reference set $\mathcal{P}$ using the score-based strategy (*cf.*, Sect. 4.1.2).
3: Ensure feasibility of all solutions in $\mathcal{P}$ using the positioning procedure (*cf.*, Sect. 4.2).
4: **repeat**
5: Identify the three best solutions $X^{(\alpha)}, X^{(\beta)}, X^{(\delta)}$ in $\mathcal{P}$.
6: **for** each solution $X^{(i)} \in \mathcal{P}$ **do**
7: Update the position using GWO equations (*cf.*, Sect. 3).
8: Restore feasibility and apply local improvement using the swapping operator (*cf.*, Sect. 4.4.2).
9: **if** $X^{(i)}$ improves **then**
10: Update $X^{(i)}$.
11: **end if**
12: **end for**
13: **if** $X^{(\alpha)}$ stagnates **then**
14: Apply SVNS to $X^{(\alpha)}$ (*cf.*, Sect. 4.5.3) to obtain X_{skew}.
15: Replace the worst solution in $\mathcal{P}$ with X_{skew}.
16: **end if**
17: **until** global stopping criterion is met
18: **return** best solution $X^{\star}$

Algorithm 1 outlines the global structure of the proposed method. The algorithm follows a two-level strategy. Population-based updates ensure global exploration (lines 5–7). Local procedures focus on feasibility and refinement (lines 3 and 8).

The process starts from a greedy initial solution (line 1). Early diversification of the reference set (line 2) improves coverage of the search space. At each iteration, the three best solutions guide the population movement (line 5). This limits random drift while maintaining diversity.

When the leading solution no longer improves (line 13), the skewed strategy is triggered. SVNS perturbs the solution in a controlled way (line 14). Structurally different solutions may be accepted, even if slightly worse. This helps the search escape local optima.

Finally, the population is updated by removing the worst solution (line 15). The best solution found during the search is returned as $X^{\star}$ (line 18). This adaptive mechanism is well suited for large and highly constrained ST2P instances.

5. Computational results

This section evaluates the performance of the proposed Skewed Population-Based Algorithm (SP-BA) by comparing its results with recent competitive methods reported in the literature, as well as with the state-of-the-art Cplex solver. The evaluation is carried out on benchmark instances taken from [20]. The proposed method and its variants were first implemented in Python and tested on the MatriCS platform[1] to tune the algorithmic parameters. After calibration, all algorithms were implemented in Python and executed under identical conditions. Experiments were conducted on a PC equipped with an Intel Pentium Core i7-8550U processor running at 1.99 GHz and 16 GB of RAM.

[1] https://www.matrics.u-picardie.fr.

TABLE 3. Characteristics of the used instances.

Type of task	Type1	Type2	Type3	Type4	Type5
$n1$	n	n	n	n	$[n/2]$
$n2$	$[n/2]$	n	$[n/2]$	n	$[n/2]$
$n12$	$[n/2]$	$[n/2]$	n	n	n

The comparative study analyzed various types of instances sourced from prior research and available in [17]. These instances were created using the generator described in [24]. They constitute a standard benchmark for evaluating exact and metaheuristic approaches on the ST2P problem.

The study considered five categories of instances, characterized by the number of tasks n, including tasks allocated to processors P_1 and P_2, as well as bi-processor tasks that require the simultaneous use of both processors. This diversity allows the assessment of algorithmic behavior under different workload distributions:

- For small instances, the number of tasks is fixed to $n = 10$. For medium instances, $n = 20$. For large-scale instances, n takes values 100, 500, and 1000, with thirty instances generated for each case. This setting ensures a balanced evaluation across different problem scales.
- The values n_1, n_2, and n_{12} represent the number of tasks assigned to processor P_1, processor P_2, and both processors, respectively. These values are generated according to the configurations reported in Table 3, where $[x]$ denotes the integer part of x. Such configurations reflect heterogeneous processor requirements commonly observed in practice.
- The processing time of task j, denoted p_j, is randomly drawn from the interval $\{1, \ldots, 50\}$. This range introduces sufficient variability while avoiding extreme task durations.
- The release date r_j of task j is randomly generated from the interval $\{1, \ldots, k\}$, where

$$k = \gamma \times \frac{s_{12} + (s_1 + s_2)}{2},$$

and $\gamma \in \{0.5, 1, 1.5\}$ controls the instance density. Lower values of γ correspond to dense instances, while higher values generate more spread-out task arrivals. Here, s_1, s_2, and s_{12} denote the total processing times of tasks assigned to P_1, P_2, and both processors, respectively.

5.1. Taguchi experimental design framework

Stochastic optimization methods, such as metaheuristics, inherently rely on probabilistic components – such as the number of global iterations, the frequency of global loop invocations, local tabu search intensity, population size, and acceptance thresholds. As a consequence, their performance is sensitive to parameter settings. Consequently, performance can vary considerably depending on the chosen configuration, highlighting the importance of robust and systematic parameter tuning strategies.

5.1.1. The main parameters

Given the number of parameters employed by the method, identifying the most influential ones is not straightforward. Parameter interactions rapidly increase the dimensionality of the search space. This reinforces the need for a structured tuning approach. To ensure both performance and reproducibility, the parameters of the proposed method were carefully tuned using a Taguchi experimental design framework. This methodology limits experimental cost while preserving statistical relevance.

Specifically, three main parameters of the proposed stochastic hybrid method were selected based on their empirical impact on performance:

- *Population size:* The number of wolves in SP-BA, affecting the breadth of global exploration. Larger populations improve diversity, while smaller ones favor faster convergence.

TABLE 4. Aggregated Taguchi experiment results by problem size (n).

Group (n)	Best Pop	Best Max Iter	Best ρ	Avg S/N (dB)	Avg CPU (s)
10	11.33	16.50	0.09	80.70	0.21
20	13.67	18.67	0.09	79.29	0.45
100	16.33	19.03	0.08	73.76	1.05
500	18.52	19.21	0.102	70.74	12.58
1000	21.54	22.11	0.1105	79.67	31.59
Mean	16.28	19.10	0.094	76.83	9.18

– *Number of global iterations:* The total number of calls to the grey wolf optimizer-based procedure, determining the global search horizon. This parameter controls the depth of the search process.
– *Skew acceptance parameter (ρ):* The tolerance threshold governing the acceptance of structurally different yet slightly inferior solutions in the adaptive skewed local search. It directly balances diversification and intensification.

These parameters were evaluated across three levels using an L9 orthogonal array in accordance with Taguchi's methodology. This design ensures a balanced exploration of the parameter space. Each configuration was executed multiple times on benchmark instances to account for randomness. Performance was measured using the relative gap $(\text{Avg Best}_{\text{SP-BA}} - \text{LB})/\text{LB}$, where LB denotes the lower bound proposed by Manaa and Chu [24]. The signal-to-noise (S/N) ratio was computed under the *smaller-is-better* assumption. The configuration with the highest average S/N ratio was selected as optimal. This procedure improves robustness, generalization, and reproducibility.

For comparison, baseline methods were either run using default parameters reported in the literature or tuned via manual grid search when such values were unavailable. The aggregated Taguchi results, grouped by problem size n, are reported in Table 4.

5.1.2. Discussions

The analysis considers the variation of three parameters: population size, maximum number of global iterations, and skew threshold ρ. Table 4 reports the averaged best values obtained for each problem size n.

– *Best population size:* The optimal population size increases steadily with problem size. It ranges from 11.33 ($n = 10$) to 21.54 ($n = 1000$). Larger instances benefit from stronger population diversity. The mean value of 16.28 supports a moderate default size, such as 20.
– *Best max iterations:* The number of iterations also increases with n, from 16.50 to 22.11. This reflects the growing complexity of larger instances. An average value close to 20 provides a robust compromise.
– *Best skew threshold (ρ):* The values vary within a narrow range, between 0.08 and 0.1105. Moderate skewing improves robustness without destabilizing convergence. The average value of 0.094 supports fixing $\rho = 0.10$.

The S/N ratio remains stable across problem sizes, with no major degradation for large instances. CPU time increases with n, as expected. This confirms the scalability of the proposed approach.

Figure 5 summarizes the influence of each parameter. Plot (a) shows that S/N improves with population size up to a threshold. Plot (b) highlights diminishing returns when increasing iterations. Plot (c) reveals the non-linear impact of the skew parameter ρ. Plot (d) confirms that CPU time grows with both n and population size.

Based on these observations, the remainder of this study adopts the following configuration: population size = 20, maximum iterations = 20, and $\rho = 0.10$. This setting provides a strong balance between solution quality and computational effort.

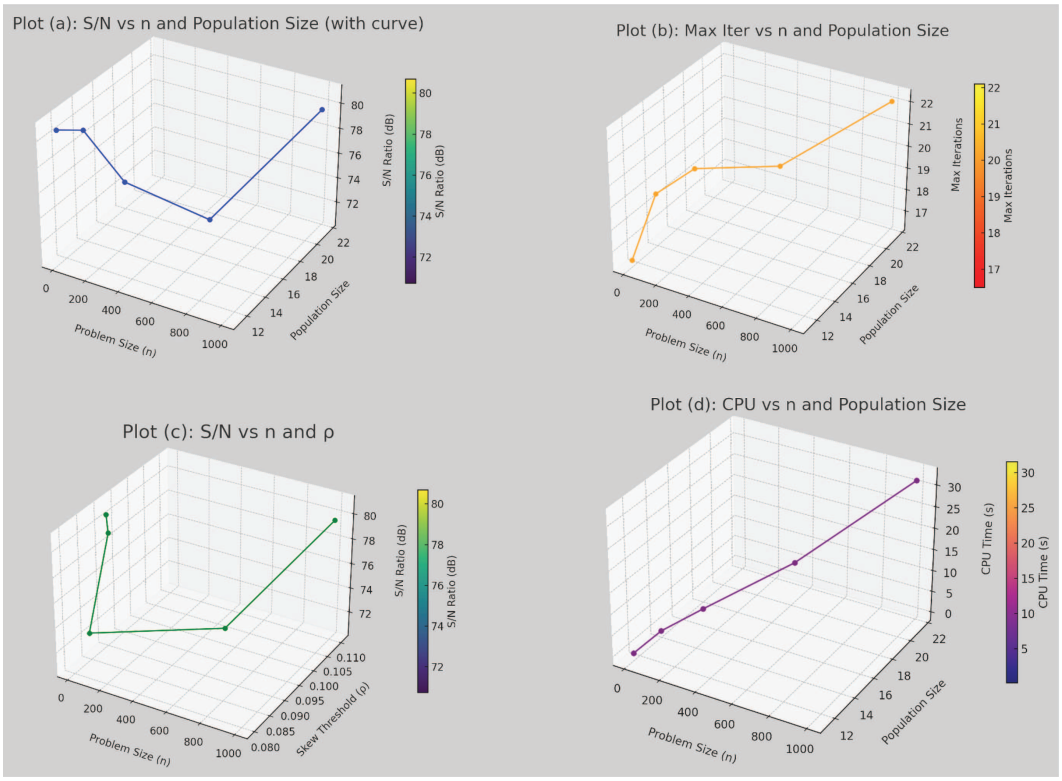

FIGURE 5. 3D visualizations of parameter influence: (a) S/N *vs.* n and Pop, (b) Max Iter *vs.* n and Pop, (c) S/N *vs.* n and ρ, (d) CPU *vs.* n and Pop.

5.2. Behavior and performance of skewed method: SP-BA

In order to confirm the parameters identified by the Taguchi method (*cf.*, Sect. 5.1), and to better understand the behavior of the proposed algorithm, a complementary experimental study is conducted. This analysis allows the method to be evaluated under different parameter settings and enables a direct comparison between its best-performing configuration and the strongest results reported in the literature.

5.2.1. Effect of the population size

The behavior of SP-BA was evaluated on instances of types 1 to 5 with $n = 20$, corresponding to medium-sized problems. This setting allows meaningful performance differences to be observed without excessive computational cost. The study focuses on the impact of population size. Starting from a minimum size of 10, the population was progressively increased in subsequent configurations. For each population size, ten independent runs were performed to ensure statistical reliability. Each run was executed within a limited runtime budget, adjusted according to the instance density γ.

Table 5 reports the results obtained by SP-BA for different population sizes. It includes instance characteristics, tight lower bounds, and both best and average solutions obtained by SP-BA$_{10}$, SP-BA$_{20}$, and SP-BA$_{30}$. Bold values indicate solutions reaching the lower bound, while italic values denote the best performance among configurations. In this case, several observations can be drawn:

– SP-BA consistently produces high-quality solutions across all tested population sizes. The gap with respect to the lower bound remains small in all cases.

TABLE 5. Behavior of SP-BA when varying the size of the population for instances with $n = 20$.

Instances		$LB_{P_{ST2P}}$	The population size: SP-BA					
			$SP\text{-}BA_{10}$		$SP\text{-}BA_{20}$		$SP\text{-}BA_{30}$	
	$n = 20$		Best	Avg	Best	Avg	Best	Avg
Type1	$\gamma = 0.5$	354.55	354.60	354.50	354.50	354.50	354.50	354.50
	$\gamma = 1$	411.60	412.40	411.90	**411.60**	**411.60**	412.40	411.90
	$\gamma = 1.5$	536.30	536.30	536.30	536.30	536.30	536.30	536.30
Type2	$\gamma = 0.5$	317.90	319.20	319.20	318.70	318.70	318.80	320.00
	$\gamma = 1$	471.00	471.10	471.10	471.10	471.10	471.10	471.10
	$\gamma = 1.5$	703.40	703.50	703.50	703.50	703.50	703.50	703.50
Type3	$\gamma = 0.5$	391.70	392.90	392.90	**392.80**	**392.80**	393.10	393.10
	$\gamma = 1$	491.70	491.90	491.90	491.80	491.80	491.90	491.90
	$\gamma = 1.5$	670.60	671.00	671.00	671.00	671.00	671.00	671.00
Type4	$\gamma = 0.5$	441.40	444.50	444.70	443.60	443.60	443.60	445.70
	$\gamma = 1$	568.40	568.50	568.60	568.50	568.50	568.50	568.60
	$\gamma = 1.5$	843.20	843.30	843.30	843.30	843.30	843.30	843.30
Type5	$\gamma = 0.5$	286.00	287.70	287.90	287.60	287.60	287.70	287.90
	$\gamma = 1$	394.90	395.20	395.20	395.10	395.10	395.20	395.70
	$\gamma = 1.5$	642.80	643.10	643.10	643.10	643.10	643.10	643.10
	Average	501.70	502.35	502.34	**502.17**	*502.17*	502.27	502.51

– Among the tested configurations, $SP\text{-}BA_{20}$ (Columns 6 and 7) achieves the best overall performance. It yields the smallest average gap (0.08) with respect to LB_{PST2P}. This version also outperforms $SP\text{-}BA_{10}$ and $SP\text{-}BA_{30}$ on average.

Overall, the results highlight the effectiveness of the $SP\text{-}BA_{20}$ configuration. This performance is primarily driven by the fact that such a setup provides a robust balance between solution quality and population diversity, ensuring the algorithm avoids premature convergence while maintaining high precision.

Supporting this observation, the results align closely with the conclusions previously drawn from the Taguchi study (*cf.*, Sect. 5.1). Given this empirical consistency across different tests, the population size is fixed to 20 for the remainder of the experimental analysis to ensure both reliability and computational efficiency.

5.2.2. Effect of the skewed strategy

To assess the skewed strategy in SP-BA, we compared the algorithm's performance across different levels of the skewing factor ρ. The goal is to assess how controlled acceptance of non-improving solutions influences the search. This mechanism enhances exploration by occasionally accepting inferior solutions, which helps prevent premature convergence.

The analysis focuses on how ρ affects solution quality and search behavior; specifically, several values of ρ were tested to study different levels of diversification. Based on Table 6:

– Moderate skewing improves performance. With $\rho = 10\%$, SP-BA reaches more than 33% of the best solutions. In contrast, $\rho = 5\%$ and $\rho = 15\%$ yield fewer best solutions, indicating insufficient exploration or excessive randomness.
– $\rho = 10\%$ balances exploration and exploitation effectively; it yields competitive bounds while maintaining stable convergence. Extreme values of ρ tend to degrade performance by either lacking diversity or introducing instability.

Average solution quality results further support these findings. As shown in Figure 6, the curve for $\rho = 10\%$ dominates, as it consistently achieves lower approximation ratios, which is favorable for minimization. The alternative configurations remain largely inferior.

TABLE 6. Effect of the skewed parameter ρ on SP-BA.

	$n = 100$	LB	$\rho = 5\%$		$\rho = 10\%$		$\rho = 15\%$	
			Av.	Best	Av.	Best	Av.	Best
Type1	$\gamma = 0.5$	3705.90	3734.90	3721.60	3710.80	3708.80	3734.60	3720.50
	$\gamma = 1$	4885.70	4890.40	4887.00	4889.30	4885.80	4934.90	4890.70
	$\gamma = 1.5$	7465.40	7472.60	7465.60	7465.80	7465.60	7465.90	7465.70
Type2	$\gamma = 0.5$	3790.60	4135.00	3967.80	4041.80	3837.30	4152.00	4006.40
	$\gamma = 1$	6113.80	6120.90	6118.50	6232.90	6117.10	6121.80	6118.80
	$\gamma = 1.5$	9374.50	9374.60	9374.60	9374.60	9374.60	9376.90	9374.60
Type3	$\gamma = 0.5$	4926.70	5144.40	5013.10	4938.30	4931.60	5152.80	5021.00
	$\gamma = 1$	6181.20	6197.60	6189.30	6198.30	6188.20	6472.60	6238.20
	$\gamma = 1.5$	9019.70	9019.90	**9019.70**	**9019.70**	**9019.70**	9066.00	**9019.70**
Type4	$\gamma = 0.5$	4977.30	5520.20	5246.80	5393.50	5058.80	5638.50	5448.90
	$\gamma = 1$	7270.70	7281.80	7276.50	7281.00	7274.30	7696.40	7397.30
	$\gamma = 1.5$	10 918.80	10 919.10	10 919.10	10 919.10	10 919.10	10 961.70	10 919.10
Type5	$\gamma = 0.5$	3853.20	4137.90	3979.10	3911.90	3863.10	4165.80	3994.60
	$\gamma = 1$	4951.80	4972.90	4955.50	5152.80	4988.50	4971.90	4962.30
	$\gamma = 1.5$	7378.20	7379.20	7378.30	7393.10	**7378.20**	7379.50	7378.80
	Average	6320.90	6420.09	6367.50	**6394.86**	**6334.05**	6486.09	6397.11

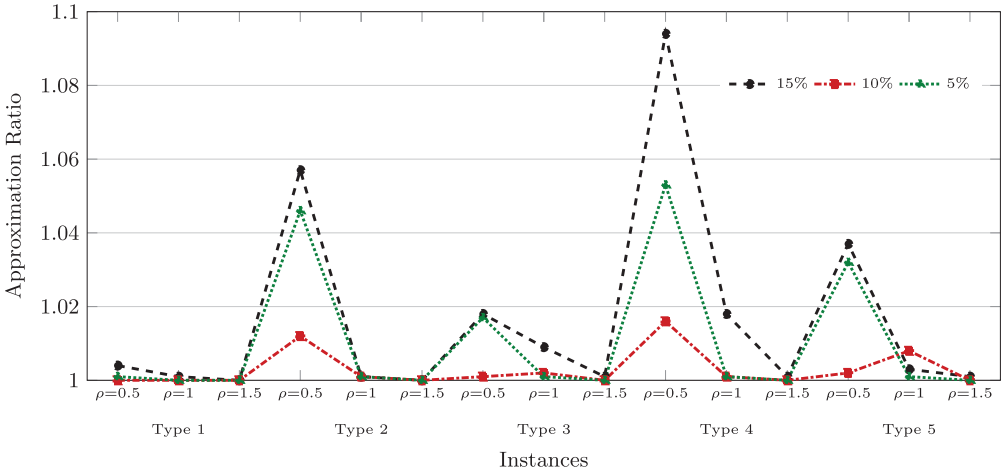

FIGURE 6. Effect of the skewed parameter ρ on the approximation ratios for instances with $n = 100$.

In summary, these results confirm that a moderate skewing level enhances exploration without harming convergence. Consequently, $\rho = 10\%$ is retained for further experiments, aligning with the recommendations of the Taguchi study (*cf.*, Sect. 5.1).

5.2.3. Impact of swap and fix-and-complete operators on optimization performance

To evaluate how neighborhood order influences VND performance, two variants are compared: VND_1 (*swap–fix-complete*) and VND_2 (*fix-complete–swap*). The fix parameter β is set to 90%, based on experiments across the range $\{80\%, 90\%, 95\%\}$, while the skewing factor ρ remains at 10%, as established in Section 5.2.2.

TABLE 7. Behavior of SP-BA regarding the order of swap and fix-complete operators.

	$n = 100$	SP-BA		Gap = V(VND_2)
		VND_1	VND_2	−V(VND_1)
Type1	$\gamma = 0.5$	3710.80	3731.30	**20.50**
	$\gamma = 1$	4889.50	4889.50	0.00
	$\gamma = 1.5$	**7465.25**	7465.80	0.55
Type2	$\gamma = 0.5$	**4041.80**	4067.20	**25.40**
	$\gamma = 1$	6233.80	**6232.90**	−0.90
	$\gamma = 1.5$	9374.60	9374.60	0.00
Type3	$\gamma = 0.5$	**4938.30**	4958.80	**20.50**
	$\gamma = 1$	**6198.30**	6200.10	1.80
	$\gamma = 1.5$	**9019.70**	9032.50	12.80
Type4	$\gamma = 0.5$	**5393.50**	5395.80	2.30
	$\gamma = 1$	7281.00	7281.00	0.00
	$\gamma = 1.5$	**10 919.10**	10 939.50	20.40
Type5	$\gamma = 0.5$	**3911.90**	3938.20	**26.30**
	$\gamma = 1$	**5152.80**	5180.10	27.30
	$\gamma = 1.5$	**7393.10**	7396.50	3.40
	Average	**6394.90**	6405.59	10.69

TABLE 8. Average performance comparison for $n = 10$, $n = 20$, and global average.

Instance size	RSBA		LA-BM		SP-BA	
	Best	Avg	Best	Avg	Best	Avg
$n = 10$	634.71	634.71	612.73	612.78	**612.72**	**612.73**
$n = 20$	516.45	528.71	502.17	502.17	**502.17**	**502.17**
Global average	575.58	581.71	557.45	557.48	**557.44**	**557.45**

The results in Table 7 highlight the performance gap between the two variants. Overall, VND_1 achieves a superior average objective value of 6394.90, reflecting an average improvement of 10.69 over VND_2. This margin is particularly significant given the problem's complexity.

While general differences appear subtle, specific instances demonstrate that neighborhood ordering significantly impacts solution quality in certain contexts. This suggests that fine-tuning operation sequences can yield incremental search improvements. Consequently, the "swap+fix" order is retained for the final SP-BA configuration.

5.3. Behavior of SP-BA versus various methods of the literature

In this section, the behavior of the proposed SP-BA is evaluated through various sets of instances, ranging from small ones to more challenging ones, as described in the characteristics of the instances (*cf.*, Tab. 3). The results are compared with those achieved by recent methods from the literature, namely RSBA [4] and LA-BM [5]. This section is divided into two parts. The first part concerns instances with $n = 10$ and $n = 20$, while the second part addresses larger instances ($n \in \{100, 1000\}$).

To improve readability, the discussion below relies on aggregated results, while detailed tables are reported in the Appendix B (Tab. B.1 for the solution values per groups, and Tab. B.2 for the CPU time per groups).

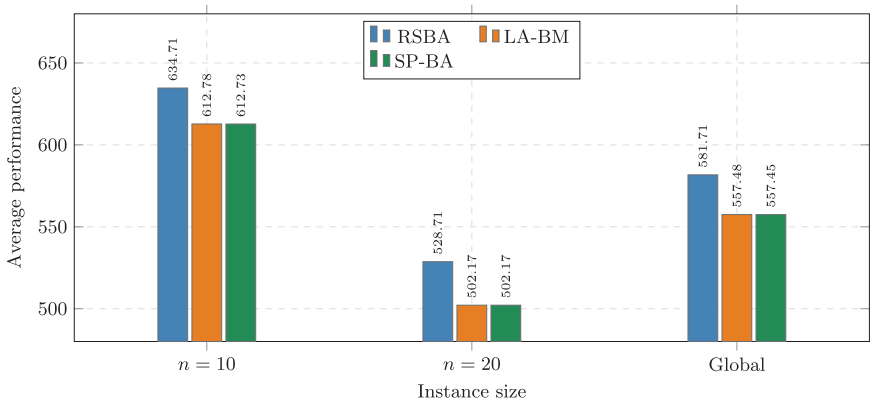

FIGURE 7. Performance comparison for $n = 10$, $n = 20$, and global average values.

5.3.1. The first set of instances: $n = 10$ and $n = 20$

Table 8 reports the average solution quality obtained by RSBA, LA-BM, and SP-BA for instances with $n = 10$ and $n = 20$, together with the corresponding global averages. Moreover, in order to facilitate the interpretation of these numerical results, Figure 7 provides a graphical comparison of the average performances for $n = 10$ and $n = 20$, allowing a direct visual assessment of the relative behavior of the three methods.
Based on the results reported in Table 8 and illustrated in Figure 7, the following observations can be made:

(1) For $n = 10$, SP-BA achieves the best average value (612.73), outperforming RSBA (634.71) and slightly improving upon LA-BM (612.78). This confirms the effectiveness of SP-BA on small instances, a trend that is clearly visible in Figure 7. Moreover, SP-BA reaches the lower bound in several cases (see Tab. B.1 in the Appendix for details).
(2) For $n = 20$, both SP-BA and LA-BM clearly dominate RSBA. SP-BA again provides the best average value (502.17), showing stable behavior on medium-sized instances. The same ranking is observed for both best and average results, as reflected by the grouped bars in Figure 7.
(3) The global averages further reinforce these observations. SP-BA yields the lowest overall values among all methods, confirming its superior solution quality across different instance sizes.

Regarding runtime, average CPU times are reported in Table 9, while detailed results across all grouped instances are available in Table B.2 of Section B.2 (Appendix B). The comparative analysis of the three methods reveals the following:

– RSBA: While this is the fastest method, it generally produces weaker solutions compared to the alternatives.
– LA-BM: Conversely, this approach requires the highest computational effort to reach a solution.
– SP-BA: On average, SP-BA is more than 50% faster than LA-BM, offering a robust compromise between computational efficiency and solution quality.

Consequently, SP-BA provides the most effective balance between solution quality and execution time. These results confirm the relevance of the parameter settings obtained through the Taguchi study and validate the practical efficiency of SP-BA on small and medium-sized instances.

Figure 8 illustrates the corresponding computational effort of the three compared methods, complementing the numerical data in Table 9. The graphical representation highlights that SP-BA maintains a significantly lower and more stable runtime profile across all instances compared to the LA-BM approach.

TABLE 9. Average CPU time (in seconds) for RSBA, LA-BM, and SP-BA.

Instance size	RSBA	LA-BM	SP-BA
$n = 10$	0.0248	0.6680	0.2638
$n = 20$	0.0065	0.7120	0.3968
Global average	0.0156	0.6900	0.3303

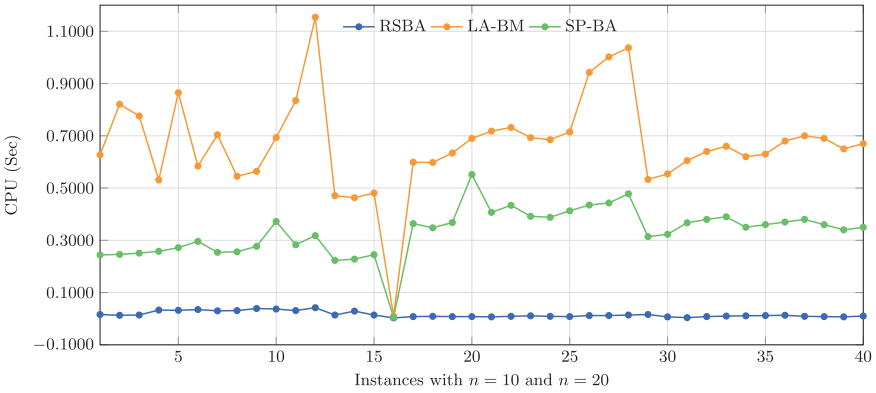

FIGURE 8. Comparison of computational effort: CPU time variations across tested instances for RSBA, LA-BM, and SP-BA.

TABLE 10. Average behavior of SP-BA versus RSBA and LA-BM.

Instance Size	LB	RSBA		LA-BM		SP-BA	
		Best	Av.	Best	Av.	Best	Av.
$n = 100$	6320.90	6712.11	6786.85	6389.79	6428.87	**6334.05**	**6394.86**
$n = 500$	31 338.23	35 133.57	35 533.14	33 655.25	33 882.07	**31 889.85**	**33 031.59**
$n = 1000$	62 545.19	71 371.67	72 295.12	68 597.39	68 917.26	**62 704.42**	**63 252.95**
Global Av.	33 401.44	37 739.12	38 205.04	36 214.14	36 409.40	**33 642.77**	**34 226.46**

5.3.2. The second set of instances: n varies between 100 *and* 1000

To evaluate the scalability and robustness of the proposed method, this section investigates a second set of benchmark instances characterized by a larger number of tasks, specifically ranging from 100 to 1000. These large-scale instances present a significantly higher combinatorial complexity, testing the ability of the algorithms to find near-optimal solutions within reasonable constraints.

Table 10 presents the bounds obtained by LB_{PST2P} (noted LB), RSBA, LA-BM, and SP-BA for instances with $n \in \{100, 500, 1000\}$. Detailed results for each configuration are provided in Table B.3 of Section B.3 (Appendix B). To evaluate these methods, performance is discussed using the average relative gap to the Lower Bound (LB):

$$Gap(\%) = \frac{V_{\text{Av}} - V_{\text{LB}}}{V_{\text{LB}}} \times 100,$$

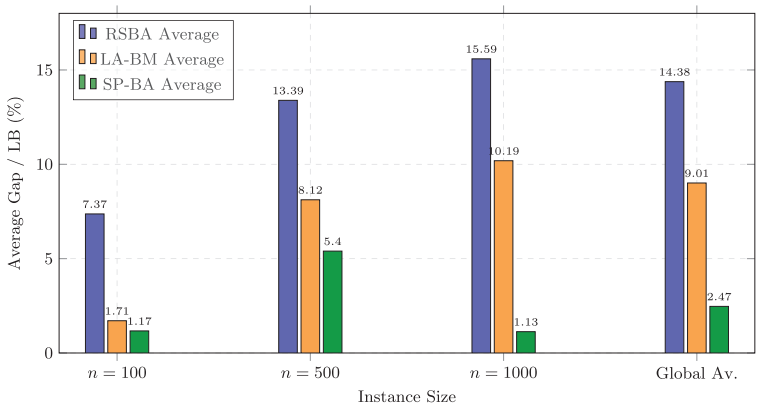

FIGURE 9. Comparison of average gaps relative to the lower bound (LB).

where V_{Av} represents the average objective value and V_{LB} is the lower bound. Consequently, lower percentage values reflect the higher solution quality and relative superiority of SP-BA. Based on the data in Table 10, the following observations can be made:

- Overall Performance: SP-BA consistently produces the best bounds across all instance sizes (as evidenced by the average values in the final two columns of Tab. 10). As shown in the summarized results, SP-BA maintains the lowest values for both best and average solutions, reflecting significant robustness regardless of instance type or scale. Moreover, Figure 9 confirms this trend, showing that SP-BA maintains a global average gap of only 2.47% relative to the LB.
- SP-BA *vs.* RSBA: The efficiency of SP-BA is particularly evident when compared to RSBA, which exhibits the highest gaps, reaching 15.59% for $n = 1000$. This confirms the superior scalability of SP-BA compared to RSBA. Specifically, SP-BA maintains a much tighter gap (1.13% for $n = 1000$) than the 15.59% reported for RSBA.
- SP-BA *vs.* LA-BM: SP-BA maintains a consistent edge over the LA-BM approach. While LA-BM performs reasonably well for $n = 100$ (1.71%), its gap increases significantly to 10.19% as the size reaches $n = 1000$. This divergence indicates that SP-BA increasingly outperforms LA-BM as the problem scale grows, ultimately achieving a global average gap of 2.47% compared to 9.01% for LA-BM.

Hence, SP-BA establishes robust performance, outperforming both RSBA and LA-BM. These results confirm the relevance of the parameter settings obtained through the Taguchi study and validate the practical efficiency of SP-BA on small and medium-sized instances.

Figure 10 illustrates the variation in computational requirements and runtime behavior for large-scale instances. An analysis of the plotted data reveals that as the problem size scales from $n = 100$ to $n = 1000$, SP-BA maintains a highly competitive computational profile:

- For $n = 100$, SP-BA remains extremely fast with an average execution time of approximately 1.6 to 2.3 s.
- As complexity increases to $n = 500$, the CPU time stays well-controlled, generally ranging between 13.9 and 32.3 s across all instance types.
- At the highest scale ($n = 1000$), although the search space expands significantly, SP-BA completes the execution within a maximum of 79.4 s, whereas RSBA remains consistently faster (around 20 to 37 s) but at the cost of solution quality as shown previously.

The stability of the CPU time across various instance types (Type1 to Type5) demonstrates the robustness of SP-BA and its ability to efficiently navigate high-dimensional scheduling search spaces without exponential growth in processing effort. Even at $n = 1000$, the processing time remains well under the 90 s threshold for all test cases.

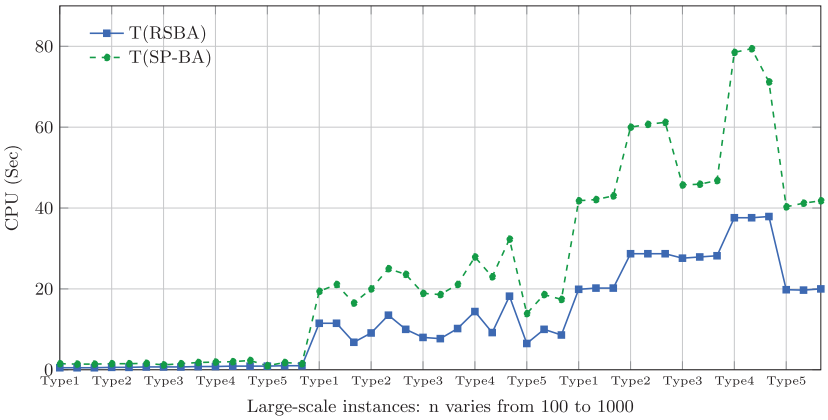

FIGURE 10. Variations in the average CPU time for all large-scale instances: SP-BA *vs.* RSBA.

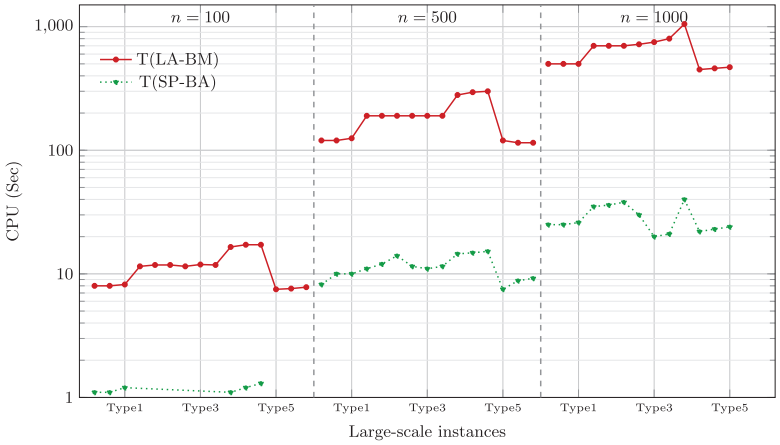

FIGURE 11. CPU comparison for large-scale instances with $n \in \{100, 500, 1000\}$.

Finally, a comparison with the computational effort of LA-BM in Figure 11 reveals the efficiency of the proposed approach. While LA-BM is highly effective for smaller instances, its execution time increases significantly on a logarithmic scale as the problem size grows. In contrast, SP-BA achieves substantially lower execution times – often by one or two orders of magnitude – while maintaining high solution quality. For instance, while LA-BM execution times climb from approximately 17 s at $n = 100$ to over 1050 s at $n = 1000$, SP-BA maintains a much flatter trajectory, requiring only 20 to 40 s for the largest instances. This represents a reduction in computational effort of over 95% in the most complex cases, reinforcing its role as a method that optimally balances computational speed and search effectiveness for large-scale scheduling.

5.3.3. Statistical analysis: large-scale instances

In order to further validate the performance of the proposed method, a statistical analysis is conducted on the upper bounds obtained by RSBA, LA-BM, and the proposed SP-BA. This analysis complements the numerical comparisons by providing formal statistical evidence. To this end, two non-parametric tests are employed: the *sign test* and the *Wilcoxon signed-rank test*. Both tests are well suited for paired comparisons and do not assume any specific distribution of the results.

TABLE 11. p-values for the *sign* and *Wilcoxon* tests on large-scale instances ($n = 100$ to $n = 1000$) at significance level $\Theta = 0.05$.

	ν_1 (RSBA *vs.* LA-BM)	ν_2 (LA-BM *vs.* SP-BA)
p-value (Sign test)	1.00	1.00
N^+	0	0
N^-	45	45
p-value (Wilcoxon)	1.00	1.00

TABLE 12. Descriptive statistics for LA-BM and SP-BA on large-scale instances ($n = 100$ to $n = 1000$).

Method	Descriptive statistics			
	Minimum	Maximum	Mean	Std. Dev.
LA-BM	3711.50	119 926.00	36 214.14	30 119.64
SP-BA	3708.80	109 893.10	33 642.77	27 229.83

The null hypothesis H_0: $\text{Algo}_1 - \text{Algo}_2 < \nu$ is defined to indicate that algorithm Algo_1 outperforms algorithm Algo_2, while the alternative hypothesis $\overline{H}_0$ states that Algo_2 performs better than Algo_1. Rejecting H_0 therefore implies a statistically significant advantage of the second algorithm.

In this study, two successive comparisons are considered. First, ν_1 compares RSBA with LA-BM. Then, the best-performing method from this comparison is evaluated against SP-BA using ν_2. This hierarchical strategy allows SP-BA to be compared against the strongest available competitor. Beyond parameter tuning, this statistical evaluation aims to confirm the robustness and superiority of SP-BA on large-scale instances.

Table 11 reports the statistical results for instances with $n = 100$, $n = 500$, and $n = 1000$. For each comparison, both the *sign test* and the *Wilcoxon signed-rank test* are reported. The p-values quantify the strength of the observed differences. More precisely, line 1 (resp. line 4) presents the p-value of the sign test (resp. Wilcoxon test), while lines 2 and 3 report the number of instances where the first algorithm dominates (N^+) or is dominated by (N^-) the second. All tests are conducted using a significance level $\Theta = 0.05$.

Several conclusions can be drawn from Table 11.

- For ν_1 (RSBA *vs.* LA-BM), both p-values are larger than the significance threshold $\Theta = 0.05$. This leads to the rejection of the null hypothesis H_0. Consequently, LA-BM is shown to significantly outperform RSBA on all large-scale instances.
- Given this result, LA-BM is then compared with SP-BA. For ν_2, both statistical tests again yield p-values equal to 1.00. This indicates a clear and systematic dominance of SP-BA over LA-BM. Moreover, SP-BA outperforms LA-BM in all 45 tested instances ($N^- = 45$), which provides strong statistical evidence of its superiority.

5.3.4. Discussion of means and standard deviation: large-scale instances

To complement the non-parametric statistical tests, a descriptive analysis of the solution distributions is conducted. Table 12 and Figure 12 provide insights based on the minimum, maximum, mean, and standard deviation of the results obtained by SP-BA and LA-BM.
Specifically:

- SP-BA achieves a lower mean of 33 642.77 compared to 36 214.14 for LA-BM, confirming its advantage in minimizing the objective function over large-scale instances.

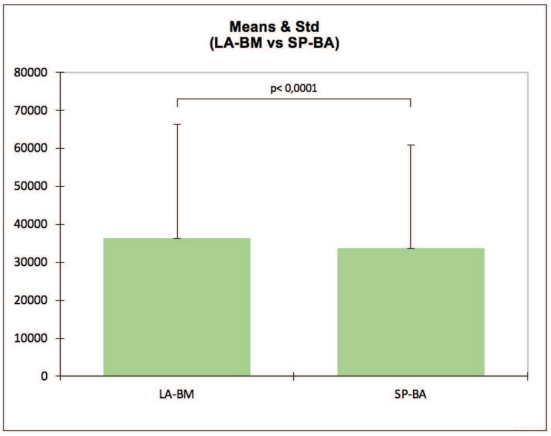

FIGURE 12. Illustration of means and standard deviations derived from descriptive statistics for LA-BM and SP-BA.

- SP-BA also achieves a slightly better minimum value (3708.80 *vs.* 3711.50 for LA-BM) and a signyficant lower maximum (109 893.10 *vs.* 119 926.00), indicating more consistent performance across the full range of instances.
- The standard deviation for SP-BA is 27 229.83, compared to 30 119.64 for LA-BM, suggesting that SP-BA shows lower variability and therefore more stable results.

Hence, these results confirm that the performance gains observed in the numerical experiments are not incidental. They are statistically significant and consistent across all large-scale instances. In contrast, LA-BM shows higher variability and produces higher values on average, which makes it less attractive in a minimization setting where both performance and consistency are preferred.

6. Conclusion

This paper investigated the impact of a skewed strategy within population-based methods to effectively tackle the task scheduling problem on two dedicated processors. The designed method, Skewed Population-based Batching Algorithm (SP-BA), integrates a modified Grey Wolf Optimizer (GWO) with a Skewed Variable Neighborhood Descent (S-VND) to achieve a robust balance between diversification and intensification. The contribution lies in the discrete adaptation of the GWO movement equations, which are coupled with a problem-specific constructive procedure to maintain schedule feasibility while exploring the combinatorial space.

The intensification phase is driven by a specialized local search that utilizes two primary neighborhood structures: a *"swap"* operator and a *"fix-and-complete"* procedure. In the fix-and-complete operator, a high percentage of the schedule (β) is preserved, allowing the algorithm to re-optimize small sub-sequences – *a technique that effectively targets local bottlenecks.* To prevent premature convergence, the search is governed by a skewing mechanism. Unlike traditional hill-climbing, which strictly accepts only improving moves, or simulated annealing and threshold-based strategies that rely on probabilistic acceptance or complex cooling schedules, this mechanism accepts non-improving solutions within a fixed, controlled distance defined by the skewing factor ρ. This enables the algorithm to escape local optima by exploring slightly inferior regions of the solution space that may lead to global minima. By maintaining a deterministic tolerance rather than a stochastic one, the search gains the flexibility to bypass local traps without the erratic trajectory often found in simulated annealing, ensuring a more stable and focused convergence toward the global optimum.

Experimental results and descriptive statistics confirm the method's superiority over recent approaches, with the Wilcoxon signed-rank test providing rigorous evidence of its robustness. The use of the Taguchi method for

parameter calibration ensured that the interaction between population size and the skewing factor was optimally tuned for scalability. Beyond theoretical performance, the proposed SP-BA framework offers significant potential for real-world applications. In manufacturing, it can be adapted for job-shop scheduling to minimize idle time, while in telecommunications, it is well-suited for dynamic packet routing. Furthermore, in project management, the strategy can optimize resource allocation across parallel workstreams to ensure critical path tasks are prioritized.

The core mechanisms —*adaptive local search, tolerance-based acceptance, and population-driven exploration*— reflect optimization strategies used in machine learning. These components make the method a promising meta-optimization tool for hyperparameter tuning and policy training. Future research will focus on extending this methodology to multi-objective scheduling and uncertain environments where task durations are stochastic. Integrating these strategies into neural architecture search and large-scale cloud-based resource allocation could further enhance scalability and generalization.

Finally, this study serves as a foundation for future research at the intersection of metaheuristic scheduling and learning-based optimization. The validated efficiency of the skewed strategy suggests that hybridizing these techniques with deep reinforcement learning could lead to even more adaptive, self-tuning optimization systems for complex, AI-driven applications.

Acknowledgments

The authors thank the anonymous reviewers for their constructive feedback during the revision process. Special appreciation is expressed to the two reviewers of the first revision and the reviewer of the second revision for their pertinent suggestions, which greatly contributed to highlighting the structure of the paper and improving both its readability and content.

Conflicts of interest

The authors declare that they have no conflict of interest.

Data availability statement

The benchmark datasets used in this study are publicly available at https://www.u-picardie.fr/eproad/, and an implementation of the proposed method can be provided upon request [17].

Author contribution statement

Fatma-Zohra Baatout (FZB), Naby Doumbouya (ND), Mhand Hifi (MH): Prof. M. Hifi: proposed and supervised the research, providing a major contribution to the work. MH: conceived the core idea of this work, centered on the design of an innovative skewed population-based method built upon a threshold-based acceptance criterion. FZB and ND: were responsible for implementing the first version of the method under the guidance of their supervisor, Professor MH, who provided the overall conceptual framework. MH: designed the final version of the method and implemented the corresponding code. MH, FZB and ND: conducted the testing of the final code version using a series of benchmark instances sourced from the literature. MH: conducted the statistical analysis, provided critical interpretation of the results, and supervised the project, leading further investigations, including sensitivity analysis of the method's parameters. All authors contributed to both the initial draft and the final version of the manuscript. The results were extensively discussed among the authors, with MH supervising the finalization of the paper. MH: carried out both main revisions of the article, in collaboration with ND, and submitted the final versions of the manuscript.

Ethical approval

This article does not involve any studies with human participants conducted by the authors.

References

[1] M. Abbasi and R. Niknam, A combined genetic algorithm and simulated annealing approach for solving competitive hub location and pricing problem. *Int. J. Appl. Manag. Sci.* **9** (2017) 88–202.

[2] M. Abbasi, N. Mokhtari, H. Shahvar and A. Mahmoudi, Application of variable neighborhood search for solving large-scale many to many hub location routing problems. *J. Adv. Manag. Res.* **16** (2019) 683–697.

[3] M. Abbasi, F. Sadough and A. Mahmoudi, Solving the fuzzy p-hub center problem using imperialist competitive algorithm. *Int. J. Mach. Learn. Cybern.* **15** (2024) 6163–6183.

[4] M. Aider, F-Z. Baatout and M. Hifi, A reactive search-based algorithm for scheduling multiprocessor tasks on two dedicated processors, in 15th FedCSIS: Conference on Computer Science and Information Systems (2020) 257–261.

[5] M. Aider, F.Z. Baatout and M. Hifi, A look-ahead strategy-based method for scheduling multiprocessor tasks on two dedicated processors. *Comput. Ind. Eng.* **158** (2021) 107388.

[6] F.Z. Baatout and M. Hifi, A two-phase hybrid evolutionary algorithm for solving the bi-objective scheduling multiprocessor tasks on two dedicated processors. *J. Heuristics* **29** (2023) 229–267.

[7] F.Z. Baatout, N. Doumbouya and M. Hifi, A hybrid population-based method for scheduling multiprocessor tasks on two dedicated processors, in IEEE, 10th International Conference on Control, Decision and Information Technologies (CoDIT) (2024) 2492–2497.

[8] L. Bianco, J. Blazewicz, P. Dell'Olmo and M. Drozdowski, Preemptive multiprocessor task scheduling with release times and time windows. *Ann. Oper. Res.* **70** (1997) 43–55.

[9] J. Blazewicz, P. Dell'Olmo, M. Drozdowski and M.G. Speranza, Scheduling multiprocessor tasks on three dedicated processors. *Inf. Process. Lett.* **41** (1992) 275–280.

[10] J. Blazewicz, P. Dell'Olmo and J. Drozdowski, Scheduling multiprocessor tasks on two parallel processors. *RAIRO Oper. Res.* **36** (2002) 37–51.

[11] O. Buffet, L. Cucu, L. Idoumghar and R. Schott, Tabu search type algorithms for the multiprocessor scheduling problem, in 10th International Conference on Artificial Intelligence and Applications (2010).

[12] Y. Bukchin, T. Raviv and I. Zaides, The consecutive multiprocessor job scheduling problem. *Eur. J. Oper. Res.* **284** (2000) 427–438.

[13] M. Drozdowski, Scheduling multiprocessor tasks: an overview. *Eur. J. Oper. Res.* **94** (1996) 215–230.

[14] F. Glover, A template for scatter search and path relinking, in European Conference on Artificial Evolution. Springer, Berlin (1997) 1–51.

[15] R.L. Graham, E.L. Lower and J.K. Lenstra, Optimization and approximation in deterministic sequencing and scheduling theory' a survey. *Ann. Discrete Math.* **5** (1979) 287–326.

[16] S. Hartmann and D. Briskorn, An updated survey of variants and extensions of the resource-constrained project scheduling problem. *Eur. J. Oper. Res.* **297** (2022) 1–14.

[17] M. Hifi, The tested instances are openly accessible to researchers in this field via. Benchmark Instances (2025). https://www.u-picardie.fr/eproad/?page_id=485.

[18] J.A. Hoogeveen, S.L. van de Velde and B. Veltman, Complexity of scheduling multiprocessor tasks with prespecified processor allocations. *Discrete Appl. Math.* **55** (1994) 259–272.

[19] A. Hosseini, A. Otto and E. Pesch, Scheduling in manufacturing with transportation: Classification and solution techniques. *Eur. J. Oper. Res.* **315** (2024) 821–843.

[20] A. Kacem and A. Dammak, A genetic algorithm to minimize the makespan on two dedicated processors, in International Conference in Control, Decision and Information Technologies (CoDIT) (2014) 400–404.

[21] A. Kononova, P. Kononovaa and A. Gordeevn, Branch-and-bound approach for optima localization in scheduling multiprocessor jobs. *Int. Trans. Oper. Res.* **27** (2020) 381–393.

[22] M. Kubale, The complexity of scheduling independent two-processor tasks on dedicated processors. *Inf. Process. Lett.* **24** (1987) 141–147.

[23] D. Lei and J. Cai, Multi-population meta-heuristics for production scheduling: a survey. *Swarm Evol. Comput.* **58** (2020) 100739.

[24] A. Manaa and C. Chu, Scheduling multiprocessor tasks to minimise the makespan on two dedicated processors. *Eur. J. Ind. Eng.* **4** (2010) 265–279.

[25] S. Mirjalili, S.M. Mirjalili and A. Lewis, Grey wolf optimizer. *Adv. Eng. Softw.* **69** (2014) 46–61.

[26] N. Mladenović and P. Hansen, Variable neighborhood search. *Comput. Oper. Res.* **24** (1997) 1097–1100.

[27] N. Mladenović, R. Todosijević, D. Urošević and M. Ratli, Solving the capacitated dispersion problem with variable neighborhood search approaches: from basic to skewed VNS. *Comput. Oper. Res.* **139** (2022) 105622.

[28] J. Mou, K. Gao, P. Duan, J. Li, A. Garg and R. Sharma, A machine learning approach for energy-efficient intelligent transportation scheduling problem in a real-world dynamic circumstances. *IEEE Trans. Intell. Transp. Syst.* **24** (2023) 15527–15539.

[29] C. Muro, R. Escobedo, L. Spector and R.P. Coppinger, Wolf-pack (canis lupus) hunting strategies emerge from simple rules in computational simulations. *Behav. Process.* **88** (2011) 192–197.

[30] A. Priya and S.K. Sahana, A survey on multiprocessor scheduling using evolutionary technique. Nanoelectronics, Circuits and Communication Systems. *Lecture Notes in Electrical Engineering*, Vol. 511. Springer, Singapore (2019) 149–160.

[31] Y. Song, L. Xing, M. Wang, Y. Yi, W. Xiang and Z. Zhang, A knowledge-based evolutionary algorithm for relay satellite system mission scheduling problem. *Comput. Ind. Eng.* **150** (2020) 106830.

[32] A. Thesen, Design and evaluation of tabu search algorithms for multiprocessor scheduling. *J. Heuristics* **4** (1998) 141–160.

[33] C. Wang, D. Deng, L. Xu and W. Wang, Resource scheduling based on deep reinforcement learning in UAV assisted emergency communication networks. *IEEE Trans. Commun.* **70** (2022) 3834–3848.

[34] A. Zahraa, A. Amiza, Al-B.M. Azmi, E. Phaklen and A.I. Hammouri, Healthcare scheduling in optimization context: a review. *Eur. J. Oper. Res.* **11** (2021) 445–469.

[35] X. Zhang, X. Li and J. Wang, Local search algorithm with path relinking for single batch-processing machine scheduling problem. *Neural Comput. Appl.* **28** (2016) 313–326.

[36] F. Zhao, L. Zhang, J. Cao and J. Tang, A cooperative water wave optimization algorithm with reinforcement learning for the distributed assembly no-idle flowshop scheduling problem. *Comput. Ind. Eng.* **153** (2021) 107082.

Appendix A. Algorithmic details

This appendix provides a detailed technical description of the core algorithmic components integrated into the proposed framework. The efficiency of the metaheuristic relies on a balanced synergy between intensification and diversification; the procedures detailed herein specify the exact mechanisms used for local refinement and search space exploration.

A.1. Basic variable neighborhood descent (VND)

In this section, the complete pseudocode of the basic Variable Neighborhood Descent (VND) procedure utilized as the intensification component within the proposed framework is reported. The algorithm systematically explores a predefined sequence of neighborhood structures, employing a first-improvement strategy to reach a high-quality local optimum.

The operational steps of Algorithm 2 are summarized as follows. Initially, the neighborhood index is set to $k = 1$ (line 1), ensuring that the search commences from the primary neighborhood structure. Subsequently, the main loop (line 2) iterates through the defined neighborhoods as long as the index does not exceed $k_{\max}$.

During each iteration, a local search is performed on the current solution X_0 using the specific neighborhood N_k (line 3), which returns an improving neighbor X_{new} if one exists. Furthermore, the algorithm evaluates whether X_{new} yields a superior objective value compared to X_0 (line 4). If an improvement is indeed detected, the current solution is updated (line 5) and the neighborhood index is immediately reset to $k = 1$ (line 6). This reset effectively forces the search to restart from the most restrictive neighborhood to ensure thorough intensification.

Conversely, if no improvement is identified within the current neighborhood, the algorithm proceeds to the next structure by incrementing the index k (line 8). Once all neighborhoods have been exhaustively explored without further gains, the loop terminates and the final solution X_0 is returned (line 11). Consequently, the resulting solution is guaranteed to be locally optimal with respect to the entire set of defined neighborhood structures.

Algorithm 2. Basic variable neighborhood descent (VND).

Input. Initial solution X_0; ordered list of neighborhood structures $\{N_1, \ldots, N_{k_{\max}}\}$.
Output. A solution X_0 locally optimal with respect to all neighborhoods.

```
1: k ← 1
2: while k ≤ k_max do
3:    X_new ← LocalSearch(X_0, N_k)
4:    if X_new improves X_0 then
5:       X_0 ← X_new
6:       k ← 1
7:    else
8:       k ← k + 1
9:    end if
10: end while
11: return X_0
```

A.2. Shaking procedure based on the fix-and-complete operator

Herein, the shaking procedure used to diversify the search within the proposed framework is described. The procedure relies on a fix-and-complete operator, which combines partial destruction and reconstruction to generate structurally different solutions while preserving feasibility.

Algorithm 3. Shaking procedure based on the fix-and-complete operator.

Input. Current solution X_0; fixing ratio β.
Output. Diversified solution X_{div}.

```
1: Start from the current solution X_0
2: Randomly fix β% of the (task, processor) pairs from X_0, producing a partial solution X_fix
3: Complete and optimize X_fix using a constructive optimization procedure, obtaining X_div
4: return X_div
```

The shaking process initiates by taking the current solution X_0 as a baseline (line 1). Subsequently, a predefined percentage β of (task, processor) assignments is randomly selected and fixed (line 2), whereas the remaining components of the solution are removed. This controlled destruction introduces necessary diversification by systematically dismantling the influence of the current local structure.

Moreover, the resulting incomplete solution X_{fix} is reconstructed and further optimized (line 3) through a constructive optimization procedure, similar to the methodology detailed in Section 4.1.1. This reconstruction phase is critical, as it restores feasibility while simultaneously facilitating the emergence of novel structural combinations that were previously unexplored.

Finally, the diversified solution X_{div} is returned (line 4) to serve as the new point of departure for subsequent intensification phases. Consequently, this transition enables the algorithm to effectively escape local optima and explore distinct regions of the search space.

A.3. Adaptive skewed variable neighborhood descent

In this section, the complete formulation of the adaptive skewed Variable Neighborhood Descent (SVND) procedure integrated into the proposed framework is detailed. While the main body of the paper provides a high-level description for brevity, this section presents the full pseudocode and a systematic explanation of its execution logic.

SVND starts by initializing the best solution X_{best} with the initial solution X (line 1). The algorithm then iterates over a fixed number of global iterations G_{iter} (line 2). At the beginning of each global iteration, the neighborhood index k and the local iteration counter ℓ are initialized (line 3). The inner loop (line 4) controls the local exploration process. A VND is applied to the current solution X (line 5), yielding a candidate solution X'. The acceptance of X' is then

Algorithm 4. Adaptive skewed variable neighborhood descent (SVND).

Input. Initial solution X; maximum number of neighborhoods $k_{\max}$; maximum number of global iterations G_{iter}; skew parameter ρ; local iteration limit $\ell_{\max}$.

Output. Best solution X_{best} found during the search.

```
1:  X_best ← X
2:  for T = 1 to G_iter do
3:     k ← 1, ℓ ← 0
4:     while k ≤ k_max and ℓ ≤ ℓ_max do
5:        X' ← VND(X)
6:        if X' improves X or z(X')/z(X) + ρ · d(X', X) > 1 then
7:           X ← X'
8:           k ← 1, ℓ ← ℓ + 1
9:           if X improves X_best then
10:             X_best ← X
11:          end if
12:       else
13:          k ← k + 1
14:       end if
15:    end while
16: end for
17: return X_best
```

evaluated using a skewed criterion (line 6), which considers both objective improvement and structural dissimilarity. If the candidate solution is accepted, the current solution is updated (line 7), the neighborhood index is reset, and the local iteration counter is incremented (line 8). Whenever X improves the best solution found so far, X_{best} is updated accordingly (lines 9–10). If the acceptance condition is not satisfied, the algorithm proceeds to the next neighborhood structure (line 13). When all neighborhoods have been explored or the local iteration limit is reached, the inner loop terminates. After completing all global iterations, the algorithm returns the best solution X_{best} (line 17).

Appendix B. Detailed experimental results

This Appendix provides a comprehensive set of experimental results that complement the analyses presented in the main body of the paper. For clarity and ease of navigation, the data is organized by problem scale and performance metric:

- Small-scale instances: Section B.1 discusses solution quality for $n = 10$ and $n = 20$ (Tab. B.1), while Section B.2 evaluates the corresponding computational effort (Tab. B.2).
- Large-scale instances: Section B.3 presents an extensive performance comparison for instances ranging from $n = 100$ to $n = 1000$ (Tab. B.3).

These results are included to ensure completeness, transparency, and to further validate the robustness of the proposed SP-BA strategy across all existing instances.

B.1. Performance analysis for small-scale instances

Table B.1 reports detailed solution quality results for small-sized instances with $n = 10$ and $n = 20$. For each instance type and each value of the skewed parameter γ, the bounds obtained by RSBA, LA-BM, and SP-BA are compared against the corresponding lower bounds (LB).

The results show that SP-BA consistently achieves the best or equal-best average values across almost all configurations.

(1) For $n = 10$, SP-BA achieves a Global Average of 612.73, matching the Best Global Average and significantly outperforming RSBA, which recorded 634.71. In specific cases such as Type 5 ($\gamma = 0.5$), SP-BA found an average value of 373.50, which is closer to the lower bound of 373.00 than LA-BM (373.60).

TABLE B.1. Effect of the skewed parameter γ on SP-BA.

	$n = 10$	LB	RSBA		LA-BM		SP-BA	
			Best	Av.	Best	Av.	Best	Av.
Type1	$\gamma = 0.5$	**400.90**	407.20	407.20	400.90	400.90	400.90	400.90
	$\gamma = 1$	**478.70**	496.40	496.40	478.70	478.70	478.70	478.70
	$\gamma = 1.5$	**789.30**	797.90	797.90	789.30	789.30	789.30	789.30
Type2	$\gamma = 0.5$	402.40	476.60	476.60	402.80	403.10	402.80	**402.80**
	$\gamma = 1$	651.00	651.10	651.10	651.10	651.10	651.10	651.10
	$\gamma = 1.5$	**914.80**	925.90	925.90	914.80	914.80	914.80	914.80
Type3	$\gamma = 0.5$	**494.90**	528.50	528.50	494.90	494.90	494.90	494.90
	$\gamma = 1$	664.00	696.60	696.60	664.10	664.10	664.10	664.10
	$\gamma = 1.5$	**924.80**	936.10	936.10	924.80	924.80	924.80	924.80
Type4	$\gamma = 0.5$	547.60	582.10	582.10	548.70	549.00	548.70	**548.70**
	$\gamma = 1$	**732.20**	781.70	781.70	732.20	**732.20**	732.20	732.30
	$\gamma = 1.5$	**674.50**	681.20	681.20	674.50	674.50	674.50	674.50
Type5	$\gamma = 0.5$	373.00	397.00	397.00	373.60	373.60	**373.50**	**373.50**
	$\gamma = 1$	**545.00**	560.80	560.80	545.00	545.20	545.00	**545.00**
	$\gamma = 1.5$	**595.50**	601.60	601.60	595.50	595.50	595.50	595.50
	Average	612.57	634.71	634.71	612.73	612.78	**612.72**	**612.73**
	$n = 20$	LB	Best	Av.	Best	Av.	Best	Av.
Type1	$\gamma = 0.5$	**354.50**	359.10	361.80	354.50	354.50	354.50	354.50
	$\gamma = 1$	**411.60**	423.80	439.50	411.60	411.60	411.60	411.60
	$\gamma = 1.5$	**536.30**	543.30	553.90	536.30	536.30	536.30	536.30
Type2	$\gamma = 0.5$	318.60	325.00	331.20	318.70	318.80	318.70	**318.70**
	$\gamma = 1$	**471.10**	486.50	500.90	471.10	471.10	471.10	471.10
	$\gamma = 1.5$	**703.50**	712.90	726.00	703.50	703.50	703.50	703.50
Type3	$\gamma = 0.5$	**392.80**	397.20	403.30	392.80	392.80	392.80	392.80
	$\gamma = 1$	**491.80**	517.90	534.20	491.80	**491.80**	491.80	491.90
	$\gamma = 1.5$	670.70	690.90	709.00	671.00	671.00	671.00	671.00
Type4	$\gamma = 0.5$	443.40	457.10	466.50	443.60	443.60	443.60	443.60
	$\gamma = 1$	**568.50**	595.80	612.80	568.50	568.50	568.50	568.50
	$\gamma = 1.5$	**843.30**	870.00	892.10	843.30	843.30	843.30	843.30
Type5	$\gamma = 0.5$	287.10	295.80	300.70	287.60	287.60	287.60	287.60
	$\gamma = 1$	**395.10**	419.00	434.30	395.10	395.10	395.10	395.10
	$\gamma = 1.5$	643.00	652.50	664.50	643.10	643.10	643.10	643.10
	Average	502.09	516.45	528.71	*502.167*	*502.173*	*502.167*	*502.173*
	Global Av.	557.33	575.58	581.71	557.45	557.48	**557.44**	**557.45**

(2) For the $n = 20$ set, the performance gap between the algorithms remains narrow but favors the proposed method; SP-BA and LA-BM both maintained a Global Average of approximately 502.17, while RSBA lagged behind at 528.71. Across both instance sizes, the Global Average for SP-BA stands at 557.45, which is superior to RSBA's 581.71 and slightly more robust than LA-BM's 557.48. These observations confirm that the proposed skewed strategy does not degrade performance on small instances and already provides a measurable advantage over competing approaches.

B.2. Discussion on results reported in Table B.2

Table B.2 presents the computational time required by the three algorithms on small-sized instances. As expected, RSBA is the fastest method due to its simpler search mechanism, achieving a Global Average runtime of only 0.0156 s. However, this speed comes at the expense of solution quality.

TABLE B.2. Illustration of the runtime for the three tested methods: RSBA LA-BM, and SP-BA.

	$n = 10$	T_{RSBA}	T_{LA-BM}	T_{SP-BA}
Type1	$\gamma = 0.5$	0.0120	0.6195	0.2379
	$\gamma = 1$	0.0106	0.8129	0.2424
	$\gamma = 1.5$	0.0105	0.7681	0.2471
Type2	$\gamma = 0.5$	0.0313	0.5253	0.2551
	$\gamma = 1$	0.0300	0.8594	0.2686
	$\gamma = 1.5$	0.0332	0.5786	0.2912
Type3	$\gamma = 0.5$	0.0279	0.6955	0.2511
	$\gamma = 1$	0.0285	0.541	0.2497
	$\gamma = 1.5$	0.0353	0.5598	0.2724
Type4	$\gamma = 0.5$	0.0354	0.6841	0.3676
	$\gamma = 1$	0.0318	0.8319	0.2793
	$\gamma = 1.5$	0.0400	1.1488	0.3122
Type5	$\gamma = 0.5$	0.0101	0.4664	0.2184
	$\gamma = 1$	0.0242	0.459	0.2224
	$\gamma = 1.5$	0.0105	0.4764	0.2415
	Average	0.0248	0.668	0.2638
	$n = 20$	T_{RSBA}	T_{LA-BM}	T_{SP-BA}
Type1	$\gamma = 0.5$	0.005	0.595	0.356889
	$\gamma = 1$	0.006	0.594	0.344172
	$\gamma = 1.5$	0.005	0.628	0.364288
Type2	$\gamma = 0.5$	0.006	0.684	0.54519
	$\gamma = 1$	0.005	0.713	0.404216
	$\gamma = 1.5$	0.006	0.730	0.429723
Type3	$\gamma = 0.5$	0.008	0.689	0.386281
	$\gamma = 1$	0.006	0.681	0.38264
	$\gamma = 1.5$	0.007	0.711	0.410399
Type4	$\gamma = 0.5$	0.008	0.940	0.432309
	$\gamma = 1$	0.009	0.998	0.437272
	$\gamma = 1.5$	0.012	1.032	0.471516
Type5	$\gamma = 0.5$	0.005	0.528	0.309444
	$\gamma = 1$	0.003	0.548	0.317277
	$\gamma = 1.5$	0.005	0.602	0.360548
	Average	0.0065	0.712	0.3968
	Global Av.	0.0156	0.6900	0.3303

(1) LA-BM reaches the highest computational cost, with an average runtime of 0.6900 s, reflecting the overhead introduced by its look-ahead mechanism. In several instances, such as Type 4 ($\gamma = 1.5$ for $n = 20$), LA-BM requires up to 1.032 s, which is significantly higher than its counterparts.
(2) SP-BA achieves a favorable compromise. While requiring slightly more time than RSBA, SP-BA remains more than twice as fast as LA-BM on average, with a Global Average of 0.3303 s compared to 0.6900 s. Specifically, for the $n = 10$ group, SP-BA averages 0.2638 s, which is a 60% reduction in computational effort compared to the 0.668 s required by LA-BM.

Even as problem complexity increases for $n = 20$, SP-BA maintains its efficiency with an average of 0.3968 s, staying well below the 0.712 s averaged by LA-BM. This confirms that the additional mechanisms embedded in SP-BA are computationally affordable and provide a well-balanced trade-off between execution speed and solution optimality.

TABLE B.3. Behavior of SP-BA versus RSBA and LA-BM.

	$n = 100$	LB	RSBA		LA-BM		SP-BA	
			Best	Av.	Best	Av.	Best	Av.
Type1	$\gamma = 0.5$	3705.9	3742.4	3748.4	3711.5	3714.3	**3708.8**	3710.8*
	$\gamma = 1$	4885.7	5167.3	5261.3	4910.9	4960.2	**4885.8**	4889.3*
	$\gamma = 1.5$	7465.4	7508.4	7595.9	7467.8	7472.2	**7465.6**	7465.8*
Type2	$\gamma = 0.5$	3790.6	4875.8	4877.1	3843.5	3865.3	**3837.3**	4041.8
	$\gamma = 1$	6113.8	6463.1	6609.1	6161.1	6231.7	**6117.1**	6232.9
	$\gamma = 1.5$	9374.5	9518.9	9621.8	9376.1	9393.7	**9374.6**	9374.6*
Type3	$\gamma = 0.5$	4926.7	5012.1	5026.5	4931.8	4934.2	**4931.6**	4938.3
	$\gamma = 1$	6181.2	6790.5	6896.2	6381.5	6460.5	**6188.2**	6198.3*
	$\gamma = 1.5$	9019.7	9134.5	9285.4	9041.7	9078.3	**9019.7**	9019.7*
Type4	$\gamma = 0.5$	4977.3	6073.3	6079.3	5101.2	5122.8	**5058.8**	5393.5
	$\gamma = 1$	7270.7	8098.7	8214.4	7587.5	7697.7	**7274.3**	7281.0*
	$\gamma = 1.5$	10 918.8	11 120.4	11 259.6	10 961.3	11 035.0	**10 919.1**	10 919.1*
Type5	$\gamma = 0.5$	3853.2	4383.8	4386.2	3866.2	3871.7	**3863.1**	3911.9
	$\gamma = 1$	4951.8	5383.8	5468.5	5124.8	5200.1	**4988.5**	5152.8
	$\gamma = 1.5$	7378.2	7408.7	7473.1	7379.9	7395.3	**7378.2**	7393.1*
	Average	6320.90	6 712.11	6786.85	6389.79	6428.87	**6334.05**	**6394.86**
	$n = 500$	LB	Best	Av.	Best	Av.	Best	Av.
Type1	$\gamma = 0.5$	18 453.2	18 937.3	19 131.9	18 526.6	18 552.1	**18 467.7**	18 612.0
	$\gamma = 1$	24 451.5	28 185.7	28 520.6	26 943.8	27 204.9	**25 491.1**	26 759.4*
	$\gamma = 1.5$	36 495.0	39 208.7	39 587.5	38 409.4	38 705.0	**36 732.6**	37 174.3*
Type2	$\gamma = 0.5$	18 654.3	24 866.9	24 980.3	20 460.4	20 544.7	**19 038.2**	19 888.6*
	$\gamma = 1$	30 397.7	34 720.3	35 304.6	33 732.9	33 906.8	**31 426.5**	32 992.0*
	$\gamma = 1.5$	45 492.8	48 927.1	49 443.7	48 096.1	48 469.1	**45 607.3**	46 177.3*
Type3	$\gamma = 0.5$	24 402.3	24 992.1	25 184.4	24 484.3	24 511.5	**24 424.1**	24 848.7*
	$\gamma = 1$	30 345.1	35 747.3	36 164.1	34 528.2	34 787.2	**31 842.9**	33 407.7*
	$\gamma = 1.5$	45 402.2	48 813.5	49 551.3	48 440.0	48 828.4	**45 557.7**	46 941.6*
Type4	$\gamma = 0.5$	24 724.1	30 913.9	31 003.4	25 848.5	25 998.7	**24 964.8**	26 284.5
	$\gamma = 1$	36 931.4	42 963.3	43 754.3	41 777.0	42 061.2	**38 307.6**	40 203.9*
	$\gamma = 1.5$	54 813.0	59 530.5	60 180.7	58 415.1	58 816.1	**55 503.3**	57 420.7*
Type5	$\gamma = 0.5$	18 486.5	21 434.9	21 478.7	18 794.4	18 867.4	**18 535.9**	19 143.5
	$\gamma = 1$	24 508.1	28 729.0	29 114.0	27 641.7	27 837.9	**25 652.8**	27 836.8*
	$\gamma = 1.5$	36 516.3	39 033.1	39 597.6	38 730.4	39 140.0	**36 795.2**	37 782.8*
	Average	31 338.23	35 133.57	35 533.14	33 655.25	33 882.07	**31 889.85**	**33 031.59**
	$n = 1000$	LB	Best	Av.	Best	Av.	Best	Av.
Type1	$\gamma = 0.5$	36 228.8	38 297.1	38 636.3	36 496.1	36 538.0	**36 291.5**	36 778.8
	$\gamma = 1$	48 217.5	56 590.7	57 377.3	54 988.7	55 319.9	**48 217.6**	48 224.9*
	$\gamma = 1.5$	72 630.1	79 063.8	79 949.9	78 165.9	78 705.3	**72 630.3**	72 631.0*
Type2	$\gamma = 0.5$	37 121.6	49 670.4	49 924.5	41 193.8	41 324.9	**38 182.6**	40 341.0*
	$\gamma = 1$	61 249.7	71 958.8	73 364.7	69 470.2	69 751.6	**61 250.5**	61 252.3*
	$\gamma = 1.5$	91 304.6	99 721.3	101 230.5	98 365.6	98 756.6	**91 304.7**	91 304.7*
Type3	$\gamma = 0.5$	48 632.4	50 405.5	50 853.3	48 901.7	48 972.1	**48 696.6**	49 610.8*
	$\gamma = 1$	60 643.9	71 806.2	73 023.5	70 492.7	70 889.3	**60 653.1**	60 659.8*
	$\gamma = 1.5$	91 361.4	101 323.8	102 708.1	99 826.2	100 346.5	**91 361.4**	91 361.4*
Type4	$\gamma = 0.5$	49 003.0	61 509.0	61 748.2	52 437.1	52 780.3	**49 941.9**	52 721.7*
	$\gamma = 1$	73 385.6	88 197.7	89 749.1	84 468.0	84 845.1	**73 387.9**	73 388.9*
	$\gamma = 1.5$	109 891.6	120 859.7	122 430.7	119 926.0	120 314.5	**109 893.1**	109 894.3*
Type5	$\gamma = 0.5$	36 596.5	42 858.2	43 007.3	37 804.9	37 970.2	**36 835.7**	38 691.0*
	$\gamma = 1$	48 729.4	58 294.7	59 218.3	56 715.4	57 076.4	**48 737.7**	48 749.2*
	$\gamma = 1.5$	73 181.7	80 018.1	81 205.1	79 708.6	80 168.2	**73 181.7**	73 184.4*
	Average	62 545.19	71 371.67	72 295.12	68 597.39	68 917.26	**62 704.42**	**63 252.95**
	Global Av.	33 401.44	37 739.12	38 205.04	36 214.14	36 409.40	**33 642.77**	**34 226.46**

B.3. Discussion on results displayed in Table B.3

Table B.3 provides a comprehensive comparison of RSBA, LA-BM, and SP-BA on large-scale instances with $n = 100$, $n = 500$, and $n = 1000$. The results clearly indicate that SP-BA consistently delivers the best bounds, both in terms of best and average values, across all instance types and parameter settings.

(1) For $n = 100$, SP-BA achieves an Average of 6394.86, outperforming LA-BM (6428.87) and RSBA (6786.85). In many cases, such as Type 3 ($\gamma = 1.5$), SP-BA matches the lower bound of 9019.7 exactly, whereas competing methods show higher deviations.
(2) The performance gap between SP-BA and the competing methods widens significantly as the instance size increases, highlighting the scalability of the proposed approach. For $n = 500$, the average result for SP-BA (33 031.59) is substantially lower than LA-BM (33 882.07) and RSBA (35 533.14).
(3) At the largest scale ($n = 1000$), SP-BA dominates overall. It records a Best Average of 62 704.42, which is remarkably close to the Global Lower Bound average of 62 545.19. In contrast, LA-BM and RSBA lag behind with averages of 68 597.39 and 71 371.67, respectively.

Finally, the Global Average across all large-scale tests confirms SP-BA's superiority with a value of 33 642.77, compared to 36 214.14 for LA-BM and 37 739.12 for RSBA. These results reinforce the conclusions regarding the robustness and effectiveness of the skewed strategy, particularly when navigating the vast and complex search spaces characteristic of large-scale task scheduling.