SILESIAN UNIVERSITY OF TECHNOLOGY PUBLISHING HOUSE

SCIENTIFIC PAPERS OF SILESIAN UNIVERSITY OF TECHNOLOGY 2024
ORGANIZATION AND MANAGEMENT SERIES NO. 206

AN APPROACH FOR SOLVING DIFFICULT SCHEDULING PROBLEMS

Piotr JĘDRZEJOWICZ[1], Ewa RATAJCZAK-ROPEL[2]*, Izabela WIERZBOWSKA[3]

[1] Gdynia Maritime University; p.jedrzejowicz@umg.gdynia.pl, ORCID: 0000-0001-6104-1381
[2] Gdynia Maritime University; e.ratajczak-ropel@wznj.umg.gdynia.pl, ORCID: 0000-0002-3697-6668
[3] Gdynia Maritime University; i.wierzbowska@wznj.umg.gdynia.pl, ORCID: 0000-0003-4818-4841
* Correspondence author

Purpose: The paper explores the integration of population-based methods and parallel processing techniques, particularly leveraging Apache Spark, for optimizing scheduling problems in real-world scenarios.
Design/methodology/approach: Diverse population-based strategies and various improvement algorithms showcase adaptability and scalability in handling several scheduling problems.
Findings: The approach is validated by computational experiments, proving its efficiency and scalability.
Research limitations/implications: Future research may include finding more effective improvement algorithms, and applying machine learning techniques for managing and controlling strategies, that are used for exploration and intensification of the feasible solution space.
Originality/value: The techniques outlined in the paper indicate promising directions for further study and development.
Keywords: scheduling, optimization, population-based methods, parallelisation.
Category of the paper: Research Paper.

1. Introduction

Combinatorial optimization problems play a crucial role in various fields, ranging from operations research and logistics to computer science and artificial intelligence. The essence of these problems lies in the search for an optimal solution from a finite set of possible solutions, where the feasible solutions form a discrete combinatorial structure. However, the inherent complexity and difficulty of these problems pose significant challenges for efficient solution methodologies.

Many combinatorial optimization problems, such as various scheduling problems (SP), the traveling salesman problem (TSP) and the knapsack problem, belong to the NP class. The difficulty in finding an optimal solution grows exponentially with the size of the input, making these problems computationally challenging. Some combinatorial optimization problems are even more challenging and fall into the category of NP-hard problems. These are problems for which no known polynomial-time algorithm exists unless P equals NP. NP-complete problems are a subset of NP-hard problems, and if a polynomial-time algorithm exists for any NP-complete problem, it implies a polynomial-time algorithm for all problems in NP.

The concept of NP-hardness and NP-completeness provides a theoretical framework for understanding the inherent difficulty of certain optimization problems. While algorithms exist for solving specific instances of these problems, finding a general solution algorithm remains an open challenge.

Given the computational intractability of many combinatorial optimization problems, researchers often resort to heuristic and approximation algorithms. Heuristics are rule-of-thumb strategies that may not guarantee an optimal solution but aim to find a good solution within a reasonable amount of time. Approximation algorithms provide solutions that are guaranteed to be close to the optimal solution, often with a known bound on the solution quality.

Combinatorial optimization problems exhibit a rich tapestry of complexity, with many problems residing in the realm of NP-hardness and NP-completeness. The development of efficient algorithms for solving these problems remains an active area of research, fueled by advancements in computational techniques, and algorithmic design. As technology continues to evolve, the quest for tackling the complexity and difficulty of combinatorial optimization problems persists, driving innovation and progress in optimization theory and practice.

The No Free Lunch Theorem, introduced by David Wolpert and William Macready in 1997 (Wolpert, Macready, 1997), is a powerful concept in the field of optimization. In essence, it states that no optimization algorithm can outperform random search over all possible optimization problems. This theorem challenges the notion of a one-size-fits-all algorithm and emphasizes the importance of tailoring optimization approaches to the specific characteristics of a problem. The theorem suggests that there is no universal algorithm that excels across all combinatorial optimization problems. Each problem has its own unique structure, and a successful optimization algorithm must exploit this structure to be effective. Therefore, understanding the characteristics of the problem at hand becomes paramount.

To navigate the challenges posed by the No Free Lunch Theorem, researchers and practitioners in combinatorial optimization have increasingly turned to customization and problem-specific knowledge. Rather than relying on generic algorithms, tailoring optimization approaches to the specific structure and constraints of a given problem is key.

This paper presents an exploration of optimizing scheduling problems in real-world contexts through the integration of population-based methods and parallel processing techniques utilizing Apache Spark environment. It investigates various approaches such as metaheuristic algorithms, problem-specific heuristics, and hybrid methods that combine different optimization techniques for solving some computationally hard scheduling problems. To obtain satisfactory results one needs approaches offering scalability, adaptability, parallelization, and capability of learning and evolving over time. The originality of this paper relies on designing and validating original software framework applied for solving different scheduling problems. We believe that by tackling scheduling problems one would arrive at ideas helpful for solving a variety of the combinatorial optimization problems.

The rest of the paper is organized as follows. In Section 2 a brief description of scheduling problems is given and two techniques are discussed – population-based optimization and solution space search parallelization. Section 3 describes three computationally difficult scheduling problems subsequently used as the test-bed for computational experiment. Section 4 describes the proposed parallelized population-based approach. Section 5 contains details of the proposed implementation for solving the test-bed problems. Section 6 describes computational experiment and its results. Finally, Section 7 contains conclusions and ideas for future research.

2. Literature Review

2.1. Scheduling

Scheduling problems are ubiquitous and critical across numerous industries and domains. They represent a broad class of optimization problems that involve assigning resources to activities over time, typically with the goal of optimizing one or more objectives such as minimizing total duration, maximizing efficiency, or balancing resource utilization. These problems are found in manufacturing, logistics, healthcare, project management, and many other sectors.

Diverse applications include:

- Manufacturing: In manufacturing, scheduling problems such as the Job Shop Scheduling Problem (JSSP) or the Flexible Job Shop Scheduling Problem (FJSSP) are vital. Efficient scheduling ensures optimal machine utilization, reduces waiting times, and accelerates product delivery, directly impacting production costs and customer satisfaction.

- Project Management: In project management, the Resource-Constrained Project Scheduling Problem (RCPSP) and its modifications such as the Multi-Skill Resource-Constrained Project Scheduling Problem (MS-RCPSP) are considered. They entail assigning resources to tasks while adhering to various constraints that restrict resource allocation. Effective scheduling is crucial for timely project completion, optimal resource utilization, and cost management.
- Healthcare: In healthcare, scheduling involves staff rostering, patient appointment systems, and operating room management. Effective scheduling is essential for patient care quality, reducing wait times, and maximizing healthcare provider efficiency.
- Transportation and Logistics: Scheduling is key in transportation for route planning and fleet management. It ensures timely deliveries, optimizes fuel consumption, and improves service quality. In logistics, scheduling affects warehouse operations, loading/unloading activities, and distribution strategies.
- IT and Computing: In the realm of IT, task scheduling in distributed and cloud computing environments is critical for balancing loads, optimizing computational resources, and reducing latency.

Efficient scheduling leads to cost savings, enhanced productivity, better service quality, and improved overall operational efficiency. In industries like manufacturing and transportation, it directly influences profitability and competitiveness. In social contexts, such as healthcare and public services, it significantly impacts service accessibility and quality.

The importance of solving various scheduling problems cannot be overstated. They are pivotal in optimizing operations, enhancing service quality, reducing costs, and improving overall efficiency in diverse sectors. However these problems are also NP-hard, meaning they are computationally intensive and challenging to solve, especially for large-scale instances. Therefore, as essential tools in solving complex scheduling problems, scalable and flexible metaheuristics have emerged, that balance between exploration and exploitation to find near-optimal solutions in reasonable time frames.

Popular metaheuristics for scheduling include:

- Genetic Algorithms (GA) (Sampson, 1976; Wu et al., 2004; Squires et al., 2022; Ajmal et al., 2021): Mimic the process of natural selection, effectively used in job scheduling and resource allocation.
- Simulated Annealing (SA) (Kirkpatrick et al., 1983; Elmohamed, Saleh, 1998; Dalila et al., 2023; Lin et al., 2021): Inspired by the annealing process in metallurgy, useful in solving job shop and flow shop scheduling problems.
- Tabu Search (TS) (Glover, Laguna, 1999; Amico, Trubian, 1993; Mathlouthi et al., 2021; Vela et al., 2020): Uses memory-based strategies to avoid cycling back to previously explored solutions, effective in complex scheduling environments.

- Ant Colony Optimization (ACO) (Dorigo, Di Caro, 1999; Rajendran, Ziegler, 2004; Yi et al., 2020) and Particle Swarm Optimization (PSO) (Kennedy, Eberthart, 1995; Wang et al., 2018; Dalila et al., 2023; Pradhan et al., 2022): Inspired by the behavior of ants and flocks of birds, respectively, these are used for their robustness in various scheduling problems.

The application of metaheuristics in solving scheduling problems is driven by the need to find high-quality solutions within a reasonable time frame, especially for problems that are too complex for classical optimization methods.

2.2. Population-based Methods and Parallelisation

Apart of the choice of metaheuristics used to solve a scheduling problem, the efficiency of searching for solutions may be improved by using population-based methods to expand the search space and applying parallelism to speed up the search process.

Population-based methods offer significant advantages in solving various optimization problems (Jędrzejowicz, 2020). The methods operate on a set of potential solutions (a population of solutions) simultaneously. They excel in exploring large solution spaces, handling complex constraints, and providing a balanced approach to both exploring new solution areas and exploiting known good solutions. The adaptability and scalability of these methods make them particularly suited for the dynamic and often computationally intensive nature of scheduling problems.

Parallelisation represents a significant advancement in metaheuristics (Alba et al., 2013; Coelho, Silva, 2021) and it improves the search in the following aspects:

- Handling large solution spaces: Many scheduling problems involve vast search spaces that are computationally intensive to explore. Parallelism allows simultaneous exploration of different regions of the solution space, significantly speeding up the search process.
- Improving solution quality and diversity: Parallel metaheuristics can work on multiple solutions concurrently, increasing the diversity of the solution pool. This diversity helps in avoiding local optima and improves the overall quality of the solution.
- Reducing computational time: One of the primary benefits of parallelism is the reduction in computational time. This is crucial for time-sensitive applications where quick decision-making is essential.
- Scalability: parallelism enhances the scalability of metaheuristics, enabling them to effectively solve larger and more complex scheduling problems that would be infeasible with sequential methods.

To solve scheduling problems described in this paper both population-based methods and parallelisation is used.

2.2.1. Integration of Apache Spark in Metaheuristics

Apache Spark (Apache Spark, 2024), a powerful open-source distributed computing platform supports the way parallelism is introduced in metaheuristics. Its ability to process large-scale data across clusters of computers efficiently makes it particularly suitable for enhancing metaheuristic algorithms, which are often computationally intensive and data-heavy. Spark provides an accessible and scalable framework for deploying metaheuristics across multiple nodes, allowing for simultaneous exploration of the solution space and faster convergence to optimal solutions.

Spark's primary strength lies in its distributed computing capability, enabling the partitioning of tasks across multiple nodes and facilitating in this way the implementation of parallel versions of algorithms, where multiple subpopulations evolve in parallel. Moreover, Spark's capability to handle large datasets seamlessly integrates with the data-intensive nature of many optimisation problems, enabling more effective and efficient data processing and analysis.

The synergy between Spark and metaheuristics drives significant advancements in solving some of the most challenging problems in various domains (Lu et al., 2020; Aljame et al., 2020).

3. Test-bed Problems

In this paper, the integration of Apache Spark with population-based metaheuristics is outlined to effectively parallelize the search for solutions in scheduling processes. To illustrate this integration, three well-known NP-hard problems are selected:

- **Job Shop Scheduling Problem (JSSP).** The Job Shop Scheduling Problem is a classic optimization problem in production and operations management. It involves scheduling a number of jobs on a set of machines. Each job consists of a specific sequence of operations, each of which must be processed on a specific machine for a certain period of time. The objective is to minimize the total time required to complete all jobs (known as the makespan). The challenge arises from the constraints: each machine can only handle one operation at a time, and once an operation starts, it must run to completion without interruption.

  The problem was addressed in (Belmamoune et al., 2022; Wei et al., 2022; Shieh et al., 2022; Jedrzejowicz Wierzbowska, 2023), each of these papers also contains the formal definition of the problem.
- **Flexible Job Shop Scheduling Problem (FJSP).** The Flexible Job Shop Scheduling Problem is a more complex variant of the traditional Job Shop Scheduling Problem. In FJSP, each operation of a job can be processed on more than one machine, adding an additional layer of complexity. The primary goal remains the minimization of the

total time to complete all jobs (makespan). This flexibility in machine assignment introduces additional decision-making dimensions, as it requires determining not only the sequence of operations for each job but also the optimal machine assignment for each operation.

The problem was formally described and examined in (Han, Yang, 2021; Jiang, Zhang, 2018; Nouri et al., 2017, Jedrzejowicz, Wierzbowska, 2022).

- **Multi-Skill Resource-Constrained Project Scheduling Problem (MS-RCPSP)**. The Multi-Skill Resource-Constrained Project Scheduling Problem is a complex scheduling problem that involves assigning renewable resources with varied skill sets to specific tasks in a project. Each resource represents human staff and possesses a cost rate and a unique combination of skills. Each task requires a specific set of skills to be completed. The goal is to optimize the project schedule by minimizing the total project duration, while adhering to resource availability, skill requirements and precedence constraints. This problem is challenging due to the intricate balance required between resource allocation, skill matching, and schedule optimization.

  The MS-RCPSP was proposed in (Bellenguez, Néron, 2005) and its formal definition can be found in (Bellenguez, Néron, 2005; Myszkowski et al., 2015; Myszkowski et al., 2019).

4. Proposed Parallelized Population-based Approach

The approach applied in this study is population-based system which uses parallelisation offered by Apache Spark.

4.1. System architecture

The system presented in this paper uses a population of individuals that represent solutions to the given scheduling problem. After the initial population of solutions is generated (at random or with the use of a number of heuristics), the solutions are improved by optimization heuristic algorithms. These algorithms are internal optimizing programs and are known as optimizing agents. Typically, a set of several agents is defined, with each one improving the solution in a different way.

The optimizing agents receive solutions from the population, enhance them, and reintegrate them into the population until the stopping criterion is met. The procedure varies across different problems and approaches. It is referred to as a control strategy. The overall system architecture is outlined in Figure 1.

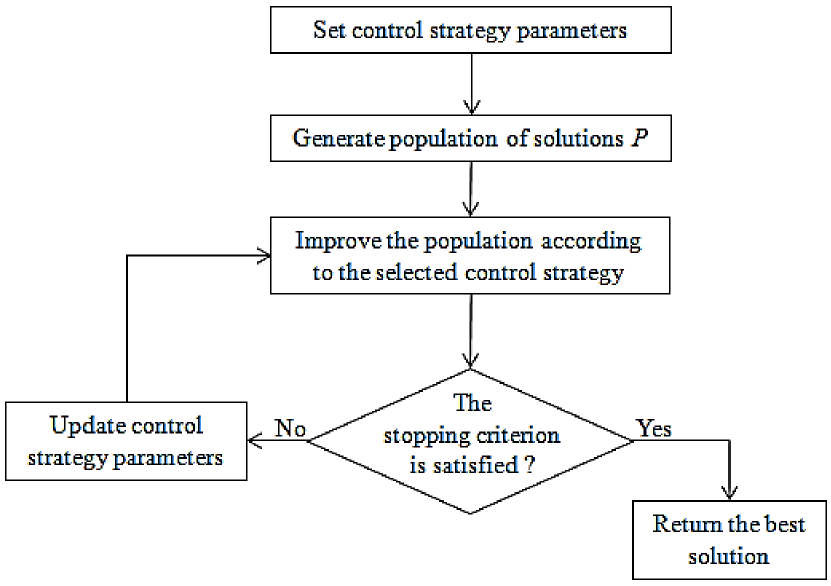
Set control strategy parameters
Generate population of solutions P
Improve the population according to the selected control strategy
The stopping criterion is satisfied ?
No
Yes
Update control strategy parameters
Return the best solution

Figure 1. Proposed system architecture schema.

Source: own study.

In the case of JSSP and FJSP problems the stopping criterion depends on the value of the objective function of the best solution in the whole population of solutions, if the value remains unchanged for a predefined number of consecutive cycles the process ends.

In the case of MS-RCPSP the average diversity in the population and the maximal number of scheduling generation schemas (SGS) procedure calls are used as the stopping criterion. An individual is represented by the sequence of activities with resources assigned. To generate a solution from the sequence, the SGS is most often used. Computations are stopped when the average diversity in the population is less than a fixed value or the number of SGS procedure calls is greater than a predefined number.

There are many control strategies and successful implementation may define these strategies in different ways. The expected result of any strategy that is used in the process is to obtain a population that over time contains better and better solutions. The crucial aspect of defining a strategy is utilisation of parallelism in execution of changes to the population.

4.2. Control strategies

The control strategies are responsible for optimization of the solutions in the population. A variety of strategies may be defined and used in the system, differing in their approach to selecting solutions, choosing heuristics for their improvement, or merging improved solutions with the population. In this study two different strategies are used.

The first strategy – SSIA (strategy based on simple improvement agents) - uses optimizing agents with relatively simple internal algorithms: they improve solutions by making a simple adjustment on the solution. Upon successful improvement, the refined solution is reintroduced back into the population, replacing the least effective solution that was initially drawn from it. The complexity of each such algorithm is very low and in one cycle of improvements hundreds or thousands such algorithms may be run. A single step of SSIA is shown in Figure 2. A predefined number of such steps is run in one cycle of optimization. This strategy is used in the case of JSSP and FJSP problems.

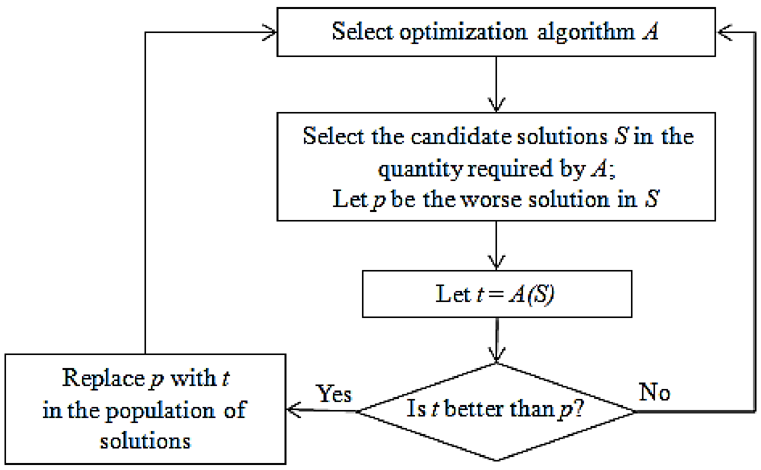
Select optimization algorithm *A*
Select the candidate solutions *S* in the quantity required by *A*;
Let *p* be the worse solution in *S*
Let *t* = *A(S)*
Is *t* better than *p*?
Yes
No
Replace *p* with *t* in the population of solutions

Figure 2. Single step in SSIA strategy.

Source: own study.

The second strategy – SRLR (strategy based on reinforcement learning rules) – is used to solve the MS-RCPSP problem. In this case the optimizing agents are slightly more complex. They represent simple metaheuristic algorithms. In one cycle of improvement a fixed number of such algorithms is run. In most cases the number does not exceed the population size. A single step of SRLR is shown in Figure 3. A predefined number of such steps is run in one cycle of optimization.

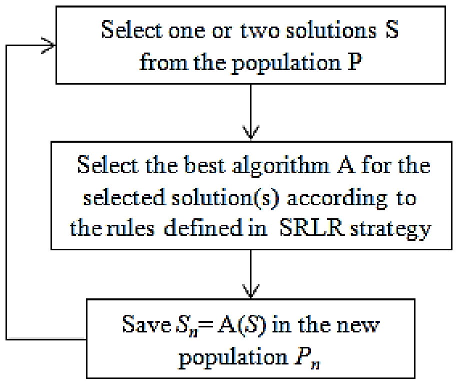
Select one or two solutions S from the population P
Select the best algorithm A for the selected solution(s) according to the rules defined in SRLR strategy
Save S_n= A(S) in the new population P_n

Figure 3. Single step in SRLR strategy.

Source: own study.

4.3. Parallelisation of the Population-based Approach

Parallelisation involves dividing the main task into smaller, independent tasks that can be executed concurrently. This concept can be applied to population-based systems as follows:

- **Division into Subpopulations:**
  - The entire population is divided into smaller subpopulations. One should recall the island-based evolutionary algorithms.
  - Each subpopulation is assigned to a separate processing unit or thread, allowing simultaneous processing.
- **Independent Evolution:**
  - Each subpopulation evolves independently.
  - By evolving separately, these subpopulations explore different regions of the solution space concurrently. Independent subpopulations are more likely to produce diverse solutions (exploration).
- **Interaction and Information Sharing:**
  - Periodically, the subpopulations are combined into one set and then divided again. In this way, new subpopulations contain solutions from various areas of the solution space.
  - Such (indirect) information sharing helps in propagating good traits across the entire population, preventing subpopulations from stagnating in local optima (exploitation).

The optimization of the populations employing the SSIA strategy is illustrated in Figure 4, while Figure 5 depicts the optimization utilizing the SRLR strategy.

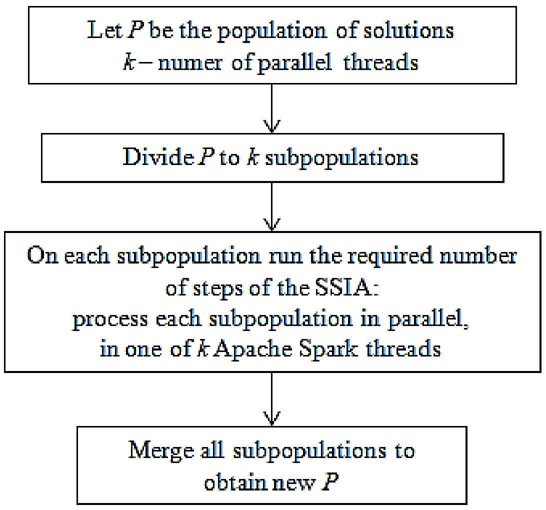
Let P be the population of solutions
k – numer of parallel threads
Divide P to k subpopulations
On each subpopulation run the required number of steps of the SSIA:
process each subpopulation in parallel,
in one of k Apache Spark threads
Merge all subpopulations to obtain new P

Figure 4. Optimization of the population with the use of SSIA.

Source: own study.

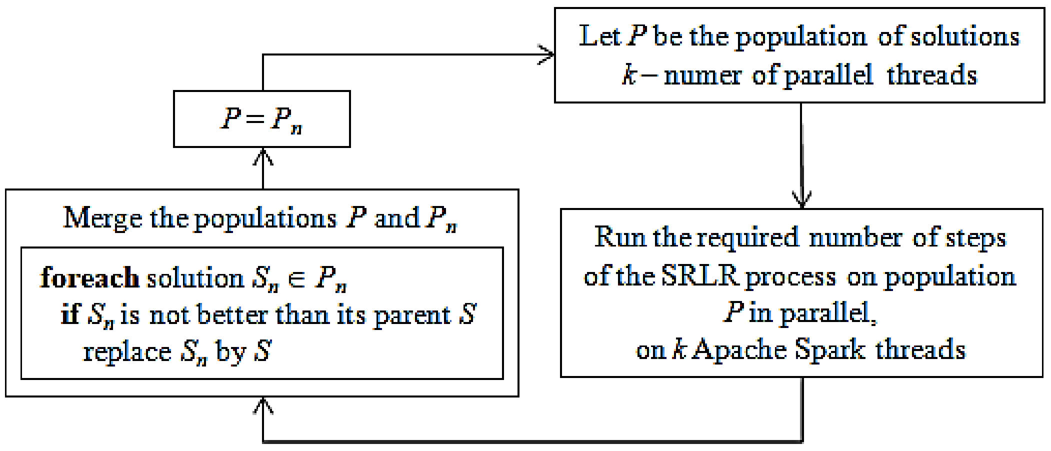
Let P be the population of solutions
k – numer of parallel threads
Run the required number of steps of the SRLR process on population P in parallel, on k Apache Spark threads
Merge the populations P and P_n
foreach solution $S_n \in P_n$
if S_n is not better than its parent S
replace S_n by S
$P = P_n$

Figure 5. Optimization of the population with the use of SEA.

Source: own study.

4.3.1. Incorporation of Apache Spark

In both the SSIA and SRLR strategies, a crucial component of the algorithm involves concurrent execution of processes in k parallel threads. This is achieved by leveraging the capabilities of Apache Spark, where each process runs independently in its dedicated thread within the Apache Spark framework.

By implementing the parallelized approach, the system can leverage the benefits of population-based methods — such as robustness and the ability to escape local optima — while significantly improving computation efficiency and solution diversity.

5. Implementation for Scheduling Problems

In the context of scheduling problems there are several elements of the system that should be defined to match the problem being solved, and these are:

- the form of solution, its structure and specific methods related to its processing, particularly the function used for calculating the objective function and functions that address constraint management,
- aforementioned methods for generating new solutions - either through randomization or by employing basic heuristic techniques,
- methods that take solution or solutions and refine them to generate improved versions (optimizing or improvement agents).

For all scheduling problems considered in this paper solutions are represented as lists.

5.1. Improvement Agents

Improvement agents are specialized heuristic or metaheuristic optimization algorithms. Each agent is designed to enhance given solution or solutions by applying specific modifications. The introduction of these agents is crucial for dynamically refining solutions, where each agent contributes uniquely to the problem-solving process. They operate on existing solutions, applying techniques like mutation, crossover, or local search, to explore the solution space more effectively and find improved solutions. The diversity and specific functions of these agents are key to addressing the complex constraints and objectives of scheduling problems.

The algorithms used in case of three considered problems are as follows:

For JSP and FJSP:

- RandomSwap – replaces elements representing jobs (or operations) on two random positions in the list that represents a solution.
- RandomMove – takes at random one element representing job (or operation) from the list that represents a solution and moves it to another, random position.
- RandomReorder – takes a random slice of the list representing a solution and shuffles the elements in this slice (the order of its elements changes at random).
- Crossover – requires two solutions. A slice from the first solution is extended with the missing elements in the order as in the second solution.

Additionaly for FJSP:

- RandomReverse – takes a random slice of the list representing a solution and reverses the order of its elements.
- PSOmove – the agent performs one movement on each available particle (solution). The movement is adapted to elements with discreet values. In the case of each particle that has to be processed there is: current solution *c* of the particle, local best solution *lB* of the particle and global best solution *gB*. The new solution is created in such a way, that each *i*-th element of the list defining the resulting solution is obtained in the following way:

$$e(i) = \begin{cases} c(i), \text{with probability } pC \\ lB(i), \text{with probability } pLb \\ gB(i), \text{with probability } pGb \end{cases}$$

where *pC*, *pLb* and *pGb* are given as parameters, *pC* + *pLb* + *pGb* = 1.

The list created in the above way may result in obtaining a solution that is not feasible. Thus, in the next step the solution’s excessive jobs/operations are removed from random positions and instead the missing jobs/operations are inserted in the same positions to create a feasible solution.

For MS-RCPSP the improving algorithms are based on simple metaheuristics. All these algorithms return the best solution found in the successive steps of the search process. Three of them use the *maxIt* parameter representing the maximum number of iterations permitted without observing any improvement. The algorithms are as follows:

- LSAm – Local Search Algorithm based on activities moving – moves activities in the solution schedule. Simultaneously, the necessary change of assigned resources is checked and performed. In one iteration all possible moves are checked and the best one is carried out.
- LSAe – similar to LSAm, exchanging pairs of activities in the solution schedule instead of moving activities.
- LSAc – Local Search Algorithm based on one-point crossover operator applied to the pair of solutions. The crossover operation can be applied in each crossing point. Hence for project with *n* activities maximum *n*-2 crossing points can be checked. Because for some projects it may be too time consuming the algorithm stops after fixed number of iteration without improvement.
- EPTA – Exact Precedence Tree Algorithm based on the concept of detecting an optimum solution by enumeration for a part of the schedule consisting of some activities. An exact solution for a part of the schedule is found. The beginning of the schedule part is selected randomly without repetition. The size of the schedule part is given as a parameter. The best solution found is remembered.
- PRA – Path-Relinking Algorithm where for a pair of solutions from the population a path between them is constructed. Next, the best of the feasible solutions from the path is selected. To construct the path of solutions the activities are moved to other possible places in the schedule. Hence the iteration number is equal to *n*-3, where *n* is the number of activities in the schedule. All possible moves are checked. Only feasible solutions are accepted. The best solution found is retained.

As one can notice a diverse array of algorithms is employed to improve solutions, illustrating the broad spectrum of complexity these algorithms can embody. On one end of this spectrum, there are straightforward, relatively simple algorithms that make incremental improvements to existing solutions. These simple algorithms are often focused on very basic modification to the solution, such as for example swapping elements within a schedule. On the other end, entire metaheuristic algorithms are utilized, which can be more complex and robust, like for example EPTA. Such algorithms do not just tweak solutions; they explore the solution space using more steps and are capable of making substantial, comprehensive improvements. Another example involves employing a single move from a well-known heuristic, such as Particle Swarm Optimization (PSO), similar to what is done in the PSOmove.

This range from simplicity to complexity in improvement algorithms allows for a versatile approach to solving the scheduling problems presented in these papers, catering to the specific needs and constraints of each problem.

5.2. Cache Memories

While solving JSSP another new feature has been used: so called cache memories.

The basic premise is that solutions are represented as lists. Agents tasked with improvement aim to enhance current solutions by altering elements of these lists through methods like moving, swapping, or adjusting parts. The incorporated cache memory serves to track and retain the location (specific index within the list that represents the solution) of each solution's most recent successful modification. This characteristic aids in concentrating the search efforts around areas close to where the last successful alteration occurred. Utilizing the data in the cache memory enhances the collaborative impact of the interactions between agents by guiding them on which segment of the solution to concentrate on in subsequent steps.

6. Computational Experiments and Results

The experiments presented in this chapter regarding individual scheduling problems are based on settings derived from the previous papers of the authors: (Jedrzejowicz Wierzbowska, 2022, 2023; Jedrzejowicz, Ratajczak-Ropel, 2023).

6.1. JSP

Experiments were run (Jedrzejowicz, Wierzbowska, 2023) on a benchmark dataset for the JSSP problem: the set of 40 instances proposed by (Lawrence, 1985), that have sizes from 5x10 to 15x15. For each task from the dataset at least 30 runs were conducted, for which the average errors and times have been calculated. The settings for the experiments may be found in (Jedrzejowicz, Wierzbowska, 2023).

The results are shown in Tables 1 and 2 under the MPF+ heading (+ stands for the cache memories from Sub-subsection 6.2.1. In both tables the results are compared with other recently published algorithms.

In Table 1 results are compared with Q-Learning Algorithm, QL, (Belmamoune et al., 2022), and a hybrid EOSMA algorithm (Wei et al., 2022) that mixes the strategies of Equilibrium Optimizer (EO) and Slime Mould Algorithm (SMA). The table shows average values calculated from average results for all tasks in considered dataset.

Table 1.
Comparison of results obtained by MPF+ with other recently published results (average error and average running time)

MPF+ (Jedrzejowicz Wierzbowska, 2023)		**QL01 (Belmamoune et al., 2022)**	**QL02 (Belmamoune et al., 2022)**	**EOSMA (Wei et al., 2022)**
avg error	**avg time [s]**	**avg error**	**avg error**	**avg error**
1.50%	143.81	5.17%	8.35%	3.20%

Note. avg err – average error calculated in reference to the best-known solution values in terms of the solution makespan.

Source: own study.

In Table 2 the results are compared with the results for the Coral Reef Optimization, CROLS, (Shieh et al., 2022). The results are calculated from results for *la* instances presented in the paper. For each instance, the results from the best model presented in (Shieh et al., 2022) is taken into account.

In both tables the errors have been calculated in reference to the best-known solution values in terms of the solution makespan.

In terms of running times, the algorithms QL0 and QL1 did not provide specific running time information. The EOSMA algorithm required between 10 and 103 seconds to execute.

Table 2.
Comparison of results obtained by MPF+ with other recently published results (average error and average running time for chosen la instances)

MPF+		**CROL1 (Shieh et al., 2022)**		**CROL2 (Shieh et al., 2022)**	
avg error	**avg time [s]**	**avg error**	**avg time [s]**	**avg error**	**avg time [s]**
1.37%	126.96	0.32%	281.99	0.39%	257.13

Note. avg err - average error calculated in reference to the best-known solution values in terms of the solution makespan.

Source: own study.

Analysis of results from Tables 1 and 2 reveals that MPF+ implementation for solving the JSSP instances performs well as compared with several other approaches, for numerous instances offering better performance or shorter computation time.

6.2. FJSP

A number of experiments was run (Jedrzejowicz, Wierzbowska, 2023) on a widely used benchmark dataset: the set of ten FJSP problems by (Brandimarte, 1993), that includes instances of the problem from the size of 10 jobs, 6 machines and 55 operations to the size of 20 jobs, 15 machines and 240 operations.

In the experiments the solutions in the initial population were drawn at random or created with the use of the metaheuristic from (Ziaee, 2014).

In Table 3 the performance of the proposed method – MPF, with no cache mamories - is compared with a number of approaches from other papers. The table presents average of the best results obtained for all problems in the Brandimarte set. By the result we understand the best value of the makespan found by the algorithm. In case of MPF and GATS+HM each Brandimarte problem was solved more than once.

Table 3.
Performance of the MPF FJSP versus other approaches

MPF		AC-SD (Han, Yang, 2021)	GWO (Jiang, Zhang, 2018)		GATS+HM (Nouri et al., 2017)	
avg makespan	avg time [s]	avg makespan*	avg makespan*	avg time [s]*	avg makespan	avg time [s]
183.7	65.8	216.9	182.2	545.0	178.3	42.26

Note. the star indicates that only one solution for each problem in the benchmark dataset was given in the corresponding paper.

Source: own study.

The results, as presented in Table 3, demonstrate both satisfactory quality and competitive computation time, making MPF a worthy addition to the set of available tools for solving FJSS problem instances.

6.3. MS-RCPSP

The computational experiment has been carried out using the benchmark instances of MS-RCPSP accessible as a part of Intelligent Multi Objective Project Scheduling Environment (iMOPSE, 2024). The test set includes 36 instances representing projects consisting from 78 to 200 activities. The detailed descriptions and benchmark data analyses can be found in (Myszkowski, 2015, 2019).

In the experiment the metaheuristics described in Section 5.2 have been used with 10 or 20 iterations. Solutions in the initial population were drawn at random or created with the use of the heuristic. Tested populations include from 30 to 50 solutions. The stopping criteria have been set as minimal average diversity in the population not greater than 0.01 and maximal number of SGS procedure calls not greater than 10000. The more detailed description of the proposed by authors approach can be found in (Jedrzejowicz, Ratajczak-Ropel, 2023).

The results for PPMHRL are shown in Tables 4 and 5. During the experiment the following results were calculated and recorded: schedule duration (makespan), standard deviation (STD) and computation time. Each problem instance has been solved 10 times and the results were averaged over these solutions.

In Table 4 the results for two considered population sizes are provided, while in Table 5, the comparison of the results from the literature.

Table 4.
Comparison of results obtained by the PPMHRL for two population sizes

PPMHRL(|P| = 30)			PPMHRL(|P| = 50)		
avg makespan	STD	avg time [s]	avg makespan	STD	avg time [s]
333.6	3.7	954.3	327.8	4	1076.4

Note. STD – standard deviation.

Source: own study.

Table 5.
Comparison of results for approaches from the literature

GRAP (Myszkowski, Siemieński, 2016)			DEGR (Myszkowski et al., 2018)			GP-HH (Lin et al., 2020)		
avg makespan	**STD**	**avg time [s]**	**avg makespan**	**STD**	**avg time [s]**	**avg makespan**	**STD**	**avg time [s]**
341.4	3.4	349.7	332.5	5.1	1494.9	320.5	320.9	988.7

Note. STD – standard deviation.

Source: own study.

PPMHRL demonstrates promising results, with the population of 50 individuals outperforming the one with 30. The average best result improves by 1.9%, AVG by 1.7%, and the STD is slightly lower. Table 5 compares the obtained results with those from the literature. The proposed approach's results are comparable to several recent papers, with one population-based algorithm, GP-HH (Lin et al., 2020), standing out and outperforming others. GP-HH achieves a better makespan value by an average of 0.7%, especially noticeable as the number of activities increases.

7. Conclusions

Making operational decision is an important managerial task. Among variety of operational problems there is a special class of computationally difficult ones. Finding optimum or satisfactory solution of a difficult problem is not an easy task, with the quality and time needed for finding such a solution could be a critical factor from the point of view of the performance, and enterprise success. It is well-known that difficult operational problems include allocation of categorical resources, routing, scheduling and other problems with a combinatorial component. Main contribution of this article is proposing and validating an approach for integrating population-based methods, and parallel computation technologies, enabling to obtain high quality solutions to difficult scheduling problems using reasonable computational resources, including reasonable computation time. We propose the adoption of parallel processing techniques, with a particular emphasis on leveraging Apache Spark for simultaneous execution of population-based metaheuristics. Apache Spark role in enhancing computational efficiency and scalability aligns seamlessly with the demands of parallelized metaheuristics. The study suggest the utilization of diverse population-based strategies and incorporating various improvement algorithms within the population ensuring adaptability and scalability in handling intricate scheduling complexities. The proposed framework has been validated experimentally showing competitive performance as compared with several state-of-the-art approaches based on various metaheuristics. It should be noted that the proposed framework can be used for solving other combinatorial optimization problems.

Future research will focus on finding more effective improvement algorithms, and on applying machine learning techniques for managing and controlling strategies for exploration and intensification of the feasible solution space.

Acknowledgements

All computations for FJSP and JSSP have been run on the Spark cluster at the Centre of Informatics Tricity Academic Supercomputer and Network (CI TASK) in Gdansk.

References

1. Ajmal, M.S., Iqbal, Z., Khan, F.Z., Ahmad, M., Ahmad, I., Gupta, B.B. (2021). Hybrid ant genetic algorithm for efficient task scheduling in cloud data centers. *Computers and electrical engineering, vol. 95*. doi: 10.1016/j.compeleceng.2021.107419
2. Alba, E., Luque, G., Nesmachnow, S. (2013). Parallel metaheuristics: recent advances and new trends. *International transactions in operational research, 20(1)*, pp. 1-48, doi: 10.1111/j.1475-3995.2012.00862.x
3. Aljame, M., Ahmad, I., Alfailakawi, M. (2020). Apache spark implementation of whale optimization algorithm. *Cluster computing, vol. 23, 09*. doi: 10.1007/s10586-020-03162-7
4. *Apache Spark*. Retrieved from: https://spark.apache.org/, 20.02.2024.
5. Bellenguez, O., Néron, E. (2005). Lower bounds for the multi-skill project scheduling problem with hierarchical levels of skills. In: E. Burke, M. Trick (Eds.), *Practice and theory of automated timetabling* (pp. 229-243). Berlin/Heidelberg: Springer.
6. Belmamoune, M.A., Ghomri, L., Yahouni, Z. (2022). Solving a job shop scheduling problem using Q-learning algorithm. In: T. Borangiu, D. Trentesaux, P. Leitão (Eds.), *Service oriented, holonic and multi-agent manufacturing systems for industry of the future* (pp. 196-209). Cham: Springer International Publishing, doi: 10.1007/978-3-031-24291-5_16
7. Brandimarte, P. (1993). Routing and scheduling in a flexible job shop by tabu search. *Annals of operations research, Vol. 41*, pp.157-183, doi: 10.1007/BF02023073
8. Coelho, P., Silva, C. (2021). Parallel metaheuristics for shop scheduling: enabling Industry 4.0. *Procedia computer science, Vol. 180*, pp. 778-786, doi: 10.1016/j.procs.2021.01.328
9. Dell'Amico, M., Trubian, M. (1993). Applying tabu search to the job-shop scheduling problem. *Annals of operations research, Vol. 41*, pp. 231-252, doi: 10.1007/BF02023076

10. Dorigo, M., Di Caro, G. (1999). *Ant colony optimization: A new meta-heuristic. Vol. 2,* doi: 10.1109/CEC.1999.782657.16
11. Elmohamed, M.A.S., Coddington, P., Fox, G. (1998). A comparison of annealing techniques for academic course scheduling. In: E. Burke, M. Carter (Eds.) *Practice and theory of automated timetabling II,* pp. 92-112. Berlin/Heidelberg: Springer.
12. Fontes, D.B., Homayouni, S.M., Gonçalves, J.F. (2023). A hybrid particle swarm optimization and simulated annealing algorithm for the job shop scheduling problem with transport resources. *European Journal of Operational Research, Vol. 306, no. 3,* pp. 1140-1157, doi: 10.1016/j.ejor.2022.09.006
13. Glover, F., Laguna, M. (1999). *Tabu search I, Vol. 1,* doi: 10.1287/ijoc.1.3.190
14. Han, B., Yang, J.J. (2021). A deep reinforcement learning based solution for flexible job shop scheduling problem. *International journal of simulation modelling, Vol. 20,* pp. 375-386, doi: 10.2507/IJSIMM20-2-CO7
15. *Intelligent Multi Objective Project Scheduling Environment (iMOPSE).* Project homepage. http://imopse.ii.pwr.wroc.pl/ 20.02.2024
16. Jedrzejowicz, P. (2019). Current trends in the population-based optimization. In: N.T. Nguyen, R. Chbeir, E. Exposito, P. Aniorté, B. Trawi ́nski (Eds.), *Computational collective intelligence* (pp. 523-534). Cham: Springer International Publishing.
17. Jedrzejowicz, P., Ratajczak-Ropel, E. (2023). *Parallelized population-based multi-heuristic system with reinforcement learning for solving multi-skill resource-constrained project scheduling problem with hierarchical skills*, pp. 243-250, doi: 10.115439/2023F2826
18. Jedrzejowicz, P., Wierzbowska, I. (2022). Implementation of the mushroom picking framework for solving flexible job shop scheduling problems in parallel. *Procedia Comput. Sci., Vol. 207(C),* pp. 292-298, doi: 10.1016/j.procs.2022.09.062
19. Jedrzejowicz, P., Wierzbowska, I. (2023). *Mushroom picking framework with cache memories for solving job shop scheduling problem,* pp. 157-164. doi: 10.15439/2023F9294
20. Jiang, T., Zhang, C. (2018). Application of grey wolf optimization for solving combinatorial problems: Job shop and flexible job shop scheduling cases. *IEEE Access, Vol. 6,* pp. 26231-26240, doi: 10.1109/ACCESS.2018.2833552
21. Kennedy, J., Eberhart, R. (1995). Particle swarm optimization. *Proceedings of ICNN'95 - International Conference on Neural Networks, Vol. 4,* pp. 1942-1948, doi: 10.1109/ICNN.1995.488968
22. Kirkpatrick, S., Gelatt, C., Vecchi, M. (1983). *Optimization by simulated annealing. Science, Vol. 220.* New York, pp. 671-80, doi: 10.1126/science.220.4598.671
23. Lawrence, S. (1984). *Resource constrained project scheduling* - technical report. Pittsburgh, PA, USA: Carnegie-Mellon University.

24. Lin, J., Zhu, L., Gao, K. (2020). A genetic programming hyper-heuristic approach for the multi-skill resource constrained project scheduling problem. *Expert systems with applications, Vol. 140,* doi: 10.1016/j.eswa.2019.112915
25. Lin, S.W., Cheng, C.Y., Pourhejazy, P., Ying, K.C. (2021). Multi-temperature simulated annealing for optimizing mixed-blocking permutation flowshop scheduling problems. *Expert systems with applications, Vol. 165,* doi: 10.1016/j.eswa.2020.113837
26. Lu, H.C., Hwang, F., Huang, Y.H. (2020). Parallel and distributed architecture of genetic algorithm on Apache Hadoop and Spark. *Applied Soft Computing, Vol. 95,* doi: 10.1016/j.asoc.2020.106497
27. Mathlouthi, I., Gendreau, M., Potvin, J.Y. (2021). A metaheuristic based on tabu search for solving a technician routing and scheduling problem. *Computers operations research, 125,* p. 105079. doi: 10.1016/j.cor.2020.105079
28. Myszkowski, P.B., Laszczyk, M., Nikulin, I. and Skowroński, M. (2019). Imopse: a library for bicriteria optimization in multi-skill resource-constrained project scheduling problem. *Soft computing, Vol. 23,* pp. 3397-3410. doi: 10.1007/s00500-017-2997-5
29. Myszkowski, P.B., P. Olech Łukasz, Laszczyk, M., Skowroński, M.E. (2018). Hybrid differential evolution and greedy algorithm (DEGR) for solving multi-skill resource-constrained project scheduling problem. *Applied soft computing, Vol. 62,* pp. 1-14. doi: 10.1016/j.asoc.2017.10.014
30. Myszkowski, P.B., Siemieński, J.J. (2016). Grasp applied to multi–skill resource–constrained project scheduling problem. In: N.T. Nguyen, L. Iliadis, Y. Manolopoulos, B. Trawiński (Eds.), *Computational collective intelligence* (pp. 402-411). Cham: Springer International Publishing.
31. Myszkowski, P.B., Skowroński, M.E., Sikora, K. (2015). A new benchmark dataset for multiskill resource-constrained project scheduling problem. *2015 federated conference on computer science and information systems (FEDCSIS),* pp. 129-138. doi: 10.15439/2015F273
32. Nouri, H.E., Belkahla Driss, O., Ghedira, K. (2017). Solving the flexible job shop problem by hybrid metaheuristics-based multiagent model. *International journal of industrial engineering, Vol. 1, 05,* pp. 1-14. doi: 10.1007/s40092-017-0204-z
33. Pradhan, A., Bisoy, S.K., Das, A. (2022). A survey on PSO based meta-heuristic scheduling mechanism in cloud computing environment. *Journal of King Saud University – computer and information sciences, Vol. 34(8, Part A),* pp. 4888-4901. doi: 10.1016/j.jksuci.2021.01.003
34. Rajendran, C., Ziegler, H. (2004). Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs. *European journal of operational research, 155(2),* pp. 426-438. Financial Risk in Open Economies. doi: 10.1016/S0377-2217(02)00908-6

35. Sampson, J. (1976). Adaptation in natural and artificial systems (John H. Holland). *Siam review, Vol. 18, 07,* doi: 10.1137/1018105
36. Shieh, C.S., Nguyen, T.T., Lin, W.W., Nguyen, D.C., Horng, M.F. (2022). Modified coral reef optimization methods for job shop scheduling problems. *Applied sciences, Vol. 12(19), Sep,* p. 9867. doi: 10.3390/app12199867
37. Squires, M., Tao, X., Elangovan, S., Gururajan, R., Zhou, X., Acharya, U.R. (2022). A novel genetic algorithm based system for the scheduling of medical treatments. *Expert systems with applications, Vol. 195,* p. 116464. doi: 10.1016/j.eswa.2021.116464
38. Vela, C.R., Afsar, S., Palacios, J.J., González-Rodríguez, I., Puente, J. (2020). Evolutionary tabu search for flexible due-date satisfaction in fuzzy job shop scheduling. *Computers operations research, Vol. 119,* p.104931. doi: 10.1016/j.cor.2020.104931
39. Wang, D., Tan, D., Liu, L. (2018). Particle swarm optimization algorithm: an overview. *Soft computing, Vol. 22, 01,* doi: 10.1007/s00500-016-2474-6
40. Wei, Y., Othman, Z., Mohd Daud, K., Yin, S., Luo, Q. (2022). Equilibrium optimizer and slime mould algorithm with variable neighborhood search for job shop scheduling problem. *Mathematics, Vol. 10, 11,* p. 4063. doi: 10.3390/math10214063
41. Wolpert, D., Macready, W. (1997) No free lunch theorems for optimization. *IEEE transactions on evolutionary computation, Vol. 1(1),* pp. 67-82. doi: 10.1109/4235.585893
42. Wu, A., Yu, H., Jin, S., Lin, K.C., Schiavone, G. (2004) An incremental genetic algorithm approach to multiprocessor scheduling. *IEEE transactions on parallel and distributed systems, Vol. 15(9),* pp. 824-834. doi: 10.1109/TPDS.2004.38
43. Yi, N., Xu, J., Yan, L., Huang, L. (2020) Task optimization and scheduling of distributed cyberphysical system based on improved ant colony algorithm. *Future generation computersystems, Vol. 109,* pp. 134-148. doi: 10.1016/j.future.2020.03.051
44. Ziaee, M. (2014) A heuristic algorithm for solving flexible job shop scheduling problem. *The international journal of advanced manufacturing technology, Vol. 71, 03,* pp. 519-528. doi: 10.1007/s00170-013-5510-z