Communication Papers of the 18th Conference on Computer Science and Intelligence Systems pp. 237–244
DOI: 10.15439/2023F2826
ISSN 2300-5963 ACSIS, Vol. 37

Parallelized Population-Based Multi-Heuristic System with Reinforcement Learning for Solving Multi-Skill Resource-Constrained Project Scheduling Problem with Hierarchical Skills

Piotr Jędrzejowicz
0000-0001-6104-1381
Gdynia Maritime University
ul. Morska 83, 81-225 Gdynia, Poland
Email: p.jedrzejowicz@umg.edu.pl

Ewa Ratajczak-Ropel
0000-0002-3697-6668
Gdynia Maritime University
ul. Morska 83, 81-225 Gdynia, Poland
Email: e.ratajczak-ropel@wznj.umg.edu.pl

***Abstract*—In this paper the Parallelized Population-based Multi-Heuristic System controlled by the Reinforcement Learning based strategy is proposed to solve the Multi-Skill Resource Constrained Scheduling Problem with Hierarchical Skills, denoted as MS-RCPSP. It is an extension of the classical RCPSP where some given pool of skills has been assigned to the resources. The MS-RCPSP as well as the RCPSP belong to the class of strongly NP-hard optimization problems. To solve the MS-RCPSP the approach consisting of evolving a population of solutions and using a set of several heuristic algorithms controlled by the reinforcement learning strategy, and executed in parallel, has been proposed. To implement the system and take advantage of the speed-up offered by the parallel computations the Apache Spark platform has been used. The system has been tested experimentally using benchmark problem instances from the iMOPSE dataset with the makespan as the optimization criterion. The proposed approach produces good quality solutions often outperforming the existing approaches.**

I. Introduction

RESOURCE MANAGEMENT plays an important role in different domains. It involves planning, scheduling, and allocating various resources such as machines, technology, money, people, or teams to a project. In a majority of organizations, the task of determining some schedules occurs regularly, often daily. Deciding on schedules requires the allocation of resources which, usually, are limited and not freely available. In project management, there are three basic types of constraints imposed on the availability of resources. These are time, cost, and scope constraints. Therefore, effectively utilizing scarce resources is important for the success of any project. Extensive research in project management has led to the proposal of different models and methods aimed at optimizing resource utilization to achieve project goals. Among several possible project management problem formulations the best known and intensively researched is the Resource Constrained Project Scheduling Problem (RCPSP) and its numerous extensions. In recent years a high amount of papers have reported on various methods of solving the RCPSP and its variants. Extensive reviews of this research effort can be found in [5], [6]. Among possible RCPSP extensions focusing on the use of human resources is the idea of considering problems where to complete a project various skills on the part of human resources are needed. The idea of considering the multi-skill resource-constrained problems has been motivated by the practical needs of projects where staff with different skills is required and needs to be scheduled and assigned. In the MS-RCPSP human resources are considered each possessing a particular set of skills, which can be applied to these activities in the project that require such skills. The primary Multi-Skill Resource Constrained Project Scheduling Problem (MSRCPSP) with skillsets has been introduced in [24] and next considered, for example in [3], [17], [1]. The most recent classification for the MSRCPSP and its extensions can be found in [27]. One of such extension is MSRCPSP with hierarchical skills proposed in [2] and commonly denoted in the literature as MS-RCPSP. It is based on both the classical RCPSP and the Multi-Purpose Machine Model Problem to find a schedule that optimizes a performance criterion like, for example the project duration i.e. makespan.

Both, the MSRCPSP and MS-RCPSP as the generalizations of RCPSP belong to the class of strongly NP-hard optimization problems [4]. Hence, most of the approaches in the literature consider applying metaheuristic algorithms. Example successful approaches include Ant Colony Optimization [20], Greedy Randomized Adaptive Search Procedure [21], [22] Teaching–learning–based optimization algorithm [28], Differential Evolution and Greedy Algorithm [22], Genetic Programming [16], Genetic Programming Hyper-Heuristic [16]. In [19] the bicriteria MS-RCPSP optimization variant was proposed including project duration and cost. In [23] a new benchmark dataset was made available for public use. The approaches proposed and made available in [19], [23] involved a Greedy Algorithm that optimizes schedule duration and a Greedy guided search controlled by a Genetic Algorithm, for minimizing schedule duration and cost [23]. The Decomposition-Based Multi-Objective Genetic Programming Hyper-Heuristic

Thematic track: Computational Optimization

has been proposed in [29].

Heuristic and meta-heuristic approaches have been important and intensively expanding area of research and development for many years. With the emergence of advanced technologies, the multi-heuristics and hyper-heuristics are commonly used in various fields, including optimization problems, search algorithms, scheduling, routing, and more generally various artificial intelligence applications. They often involve selecting, combining, or switching between different heuristics based on certain conditions, problem characteristics, or performance metrics. They are also used in solving project scheduling problems [15], [18], [25], [29]. The idea behind a multi-heuristic approach is that different heuristics may be more effective or efficient in different parts of a problem solution space. By employing a combination of heuristics, the approach can exploit their complementary strengths and mitigate their individual weaknesses. This can lead to improved problem-solving performance, such as faster convergence to a solution, better quality solutions, or increased robustness to different problem instances.

Multi-heuristic approaches can be more efficient using parallel computations which are commonly used in optimisation. They provide significant advantages in terms of speed, scalability, solution space exploration, and handling complex problem structures. By leveraging multiple processing units or distributed computing resources, parallel algorithms can efficiently solve optimization problems, leading to an improved performance, faster convergence, and better quality solutions. One of the tools for parallel computing is Apache Spark. Apache Spark is an open-source framework for processing big data in parallel across clusters or cloud architectures. Spark's core data structure is called Resilient Distributed Datasets (RDDs), which improves the performance of iterative algorithms and data mining tools. The platform automatically handles program distribution and data splitting. Spark's scheduler optimizes operations using data locality and lazy evaluation.

In this paper a Parallelized Population based Multi-Heuristic (PPMHRL) for solving MS-RCPSP is proposed, implemented and validated. The approach belongs to the population-based metaheuristics class. It is based on using four types of optimization heuristic algorithms controlled by a strategy based on Reinforcement Learning technique. The heuristic algorithms include three types of local search algorithms, the path relinking algorithm and exact solution based heuristic. To implement this approach the Scala language, Apache Spark framework and RDD collections have been used. The proposed approach has been tested experimentally using benchmark instances from the iMOPSE [30] library. The makespan minimization has been used as the optimization criterion.

The paper is constructed as follows: Section II contains the formulation of the MS-RCPSP problem. Section III provides a description of the proposed Multi-Heuristic Population Based Approach with Reinforcement Learning for solving instances of the MS-RCPSP. The section contains also a description of the optimization heuristic algorithms used: local search and path relinking. In section IV the computational experiment carried out has been described, including parameter settings experiment plan, experiment results, and comparisons of results with some other approaches. Finally, Section V contains conclusions and suggestions for future research.

II. Problem Formulation

In the paper, we consider the project management problem where activities to be executed require skills, and the available multi-skilled resources possess these skills.

The considered Multi-Skill Resource-Constrained Project Scheduling Problem with hierarchical skills can be described using classification scheme proposed in [7] for scheduling problems. An extension of this classification scheme that allows the representation of multi-skilled resource-constrained project scheduling problems and their extensions was proposed in [27] recently. The considered problem class is denoted as $ms, 1, H, TR, Flex|cpm, 1|C_{max}, C$.

In the MS-RCPSP problem the set of n activities (tasks) and m renewable resource types are considered. Each activity has to be processed without interruption to complete the project. The duration of activity a_j, $j = 1, \ldots, n$ is denoted by d_j. The types of resources represent human staff with different skills. Every resource r_k, $k = 1. \ldots, m$ possesses a subset of skills Q^k from the skill pool Q defined in a project and the salary paid for performed work as hourly rate (cost) c_k. In a given period of time, only one resource can be assigned to a given activity.

Each activity requires a set of skills to be executed denoted as Q_j, but not every resource can be applied to its realization. Each resource skill is labelled with familiarity level, that is the resource r_k is capable of performing the activity a_j only if r_k disposes skill required by a_j at the same or higher level.

There are precedence relations of the finish-start type with a zero parameter value (i.e. $FS = 0$) defined between the activities in the project. In other words activity a_j precedes activity a_i if a_i cannot start until a_j has been completed. S_j (P_j) is the set of successors (predecessors) of activity a_i, $j = 1, \ldots, n$.

The objective is to find a schedule S of the project activities finishing times $[f_i, \ldots, f_n]$, where the resource and precedence constraints are satisfied, such that the schedule duration (makespan) $MS(S) = s_n$ is minimized.

Since the MS-RCPSP is a generalization of the RCPSP, it belongs to the class of the strongly NP-hard problems [4], [19].

More detailed description and formal definition of the MS-RCPSP can be found in [2], [19], [23].

III. Parallelized Population-Based Multi-Heuristic System with RL for MS-RCPSP

A. Apache Spark based Implementation

To implement the proposed system Scala language and Apache Spark environment have been used. Apache Spark is an open-source framework designed for processing big data in parallel across clusters or cloud architectures. It prioritizes ease

of use and leverages data locality to optimize computations while maintaining the required fault tolerance. Apache Spark is currently one of the most popular and fastest distributed computing frameworks, and it stands-out as the largest open-source project in data processing.

The architecture of Spark involves a master node and multiple worker nodes. The master node handles task scheduling, resource allocation, and error management, while the worker nodes perform parallel processing of Map and Reduce tasks. The platform automatically handles program distribution and data splitting for the users.

The core data structure in Spark is called Resilient Distributed Datasets (RDDs). RDD collections enhance distributed, parallel computation of iterative algorithms and interactive data mining tools. RDDs enable using parallel data structures and parallel computing.

Spark's scheduler efficiently executes operations specified by RDDs, exploiting data locality to avoid producing unnecessary data copies between nodes. RDDs are so called lazy structures evaluated, meaning the operations are performed only when the result is requested. This allows to increase the efficiency of parallel computing. Spark's built-in constraint solver can optimize the transformation graph by eliminating certain operations. RDDs also enable efficient fault tolerance by tracking the history of transformations rather than duplicating data between nodes.

The proposed Parallelized Population-based Multi-Heuristic system controlled by Reinforcemant Learning (RL) strategy is denoted as PPMHRL. The PPMHRL uses the parallel computing capabilities of Spark in order to solve MS-RCPSP problem instances stored in a population. In this approach to use the Spark capabilities efficiently its build-in parallelization mechanism has been used. To solve the MS-RCPSP the population of solutions, optimization heuristic algorithms and control strategy have been proposed and implemented. Individuals from the population of solutions are improved by optimization heuristic algorithms controlled by the RL strategy. The proposed optimisation heuristic algorithms are described in the following subsection.

B. Optimization Heuristic Algorithms Solving MS-RCPSP

To solve the MS-RCPSP with makespan minimalization the heuristic algorithms coded in Scala language have been used. The algorithms proposed in [12] have been improved and adjusted to the new system. Hence, five kinds of optimization heuristic algorithms are used:

- `LSAm` - Local Search Algorithm based on activities moving,
- `LSAe` - Local Search Algorithm based on activities exchanging,
- `LSAc` - Local Search Algorithm based on one point crossover operation,
- `PRA` - Path Relinking Algorithm based on activities moving,
- `EPTA` - Exact Precedence Tree Algorithm.

All proposed algorithms search for feasible solutions only and feasible solutions only are stored in the population.

The above mentioned `LSA` is a simple local search algorithm which finds the local optimum by moving (`LSAm`) activities or exchanging (`LSAe`) pairs of activities in the solution schedule. Simultaneously, the necessary change of assigned resources is checked and performed. In one iteration all possible moves or exchanges are checked and the best one is carried out. The best solution found is remembered. The only parameter of these algorithms is:

- $maxIt_{LSAm}$ - the maximum number of iterations without improvement for activities moving,
- $maxIt_{LSAe}$ - the maximum number of iterations without improvement for activities exchanging.

The `LSAc` is LSA 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. The best solution found is remembered. The only parameter of this algorithm is:

- $maxIt_{LSAc}$ - the maximum number of iterations without improvement.

The `PRA` is a 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. All possible moves are checked. Only feasible solutions are accepted. The best solution found is remembered. The algorithm has no parameters.

The `EPTA` is an exact precedence tree algorithm based on the concept of finding an optimum solution by enumeration for a partition of the schedule consisting of some activities. The implementation proposed for RCPSP [9] has been adopted for solving MS-RCSP by adjusting constraints for multi hierarchical skill levels. An exact solution for a part of the schedule beginning from activity on chosen position is found. The activity position is chosen randomly without repetition. The best solution found is remembered. The algorithm has two parameters:

- $maxIt_{EPTA}$ - the maximum number of iterations without improvement,
- $nPart_{EPTA}$ - the size of schedule partition for which the exact solution is found.

C. Architecture of the PPMHRL System

The Parallelized Population-based Multi-Heuristic system controlled by Reinforcement Learning strategy (`PPMHRL`) searches for solutions of MS-RCPSP using a set of improvement heuristic algorithms. The initial population of solutions (individuals) is generated using random priority rule and serial forward SGS (Schedule Generation Scheme). An individual is represented by the sequence of activities with resources

assigned. To generate a solution from the sequence, the serial forward SGS is used. Individuals from the population are, at the following computation stages, improved by optimization heuristic algorithms described in section III-B. The behaviour of the system is controlled by the strategy. The control strategy defines parameters and methods for the whole system and is based on Reinforcement Learning.

The set of used priority rules includes ones known for RCPSP and proposed for MS-RCPSP [13], [14], [15], [12]:

- SPT - Shortest Processing Time first,
- LPT - Longest Processing Time first,
- EST - Earliest Start Time first,
- EFT - Earliest Finish Time first,
- LST - Latest Start Time last,
- LFT - Latest Finish Time last,
- HLSR - Highest Level of Skill Required first - activities are sorted by the level of skill required and SPT, which means that activities with the same level are sorted according to SPT,
- LLSR - Lowest Level of Skill required last,
- MRS - Most Required Skills first - for each skill in the project the sum of durations of activities which need this skill is calculated, next the activities are sorted by the duration of required skill and LPT,
- LRS - Least Required Skills last - for each skill in the project the sum of durations of activities which need this skill is calculated, next the activities are sorted by the duration of required skill and LPT.

To implement the approach in Spark two main RDD collections are used, one to store individuals in the population and the second one to store tuples. Each tuple contains a solutions and the algorithm that has been assigned to improve them. For selecting solutions and assigning algorithms to them the control strategy is responsible. The system state is stored and used by the control strategy to manage effectively the process of searching for the best solution. The general schema of the proposed approach can be seen in Fig. 1.

The Reinforcement Learning (RL) based cooperation strategy to control agents was proposed in [10], [11] for RCPSP and next partly adopted in the approach of solving MS-RCPSP by Asynchronous Team (A-Team) of agents in Multi-Agent System (ATMAS) described in [12]. In the approach proposed in this paper the concept of the RL based strategy has been used to control the execution of optimization heuristic algorithms using RDD collections in Spark environment. To describe the approach in a more detailed manner the following notation is used:

- P - population of $|P|$ solutions (individuals),
- S - solution,
- $nSGS$ - number of SGS procedure calls,
- $maxSGS$ - maximum number of SGS procedure calls,
- $angDiv$ - average diversity in the population P,
- $minAvgDiv$ - minimum average diversity in P,
- nS_{new} - number of newly generated solutions,
- pRA - percent of solution to be removed from population P, it is equal to the percent of the new ones to generate and add
- $selMet$ - selection method used to select the individuals to improve from P,
- $genMet$ - method generating new individuals,
- $merMet$ - method merging two populations,
- $f(S)$ - fitness function.

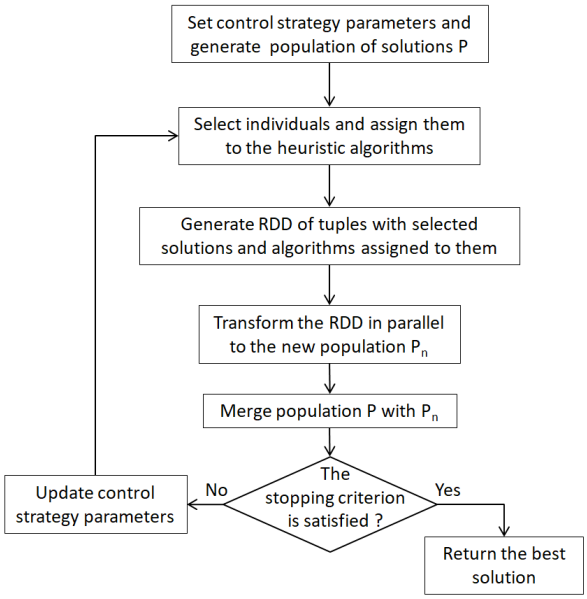

Fig. 1. Proposed PPMHRL system architecture schema

Additionally to control the system, two probability measures have been used:

- p_{mg} - probability of selecting the method $mg \in Mg$ to generate a new individual,
- p_{ma} - probability of selecting the optimisation algorithm $ma \in Ma$ used to improve individuals in P.

To generate new individuals we have proposed the following four possible methods:

- mgr - randomly,
- $mgrp$ - randomly using random priority rule,
- mgb - random changes of the best individual in P,
- mgw - random changes of the worst individual in P.

For each method the weight w_{mg} is calculated, where $mg \in Mg$, $Mg = \{mgr, mgrp, mgb, mgw\}$. The w_{mgr} and w_{mgrp} are increased when the population average diversity decreases and they are decreased in the opposite case. The w_{mgb} and w_{mgw} are decreased where the population average diversity increases and they are increased in the opposite case.

There are five optimization heuristic algorithms described above. For each of them the weight w_{ma} is calculated, where $ma \in Ma$, $Ma = \{maLSAm, maLSAe, maLSAc, maPRA, maEPTA\}$. The w_{ma} is increased if the optimization agent received the improved solution and is decreased if this not the case. Additionally, the weights for $maLSAc$ and $maPRA$ are

increased where the average diversity of the population decreases and they are decreased in the opposite case. The weights for $maLSAm$, $maLSAe$ and $maEPTA$ are increased where the average diversity of the population increases and they are decreased in the opposite case.

The probabilities of selecting the method are calculated as following:

$$p_{mg} = \frac{w_{mg}}{\sum_{mg \in M} w_{mg}}, \qquad p_{ma} = \frac{w_{ma}}{\sum_{ma \in M} w_{ma}}.$$

The parameters settings and the resulting probabilities allow to control the system behaviour and balance the exploration and exploitation processes. The p_{ma} is used in $selMet$ for selecting the optimization algorithm for individuals from the population that are subject to an intended improvement. The method is showed as Algorithm 1.

Algorithm 1 $selMet(P)$

```
generate RDD collection
arrange the solutions in P in random order
for all solutions in P do
    as = ∅
    copy to as one or two consecutive solutions from P
    if |as| == 1 then
        select ma from {maLSAm, maLSAe, maEPTA}
        with the probability p_maLSAm, p_maLSAm and
        p_maEPTA, respectively
    else
        select ma from {maLSAc, maPRA} with the prob-
        ability p_maLSAc and p_maPRA, respectively
    end if
    add the tuple (as, ma) to RDD
end for
return RDD
```

The p_{mg} is used in $merMet$ method to merge the old population with the new one created in improvement stage by RDD transformation. The pseudocode of the $merMet$ method is presented as Algorithm 2.

It can be noticed that, as a result, the better solutions received from optimization algorithms replace the worse ones in the population. Moreover, the new solutions are generated in each stage with the calculated probability according to $genMet$ presented as Algorithm 3.

All decreasing–increasing operations are performed following the proposed control strategy. As the stopping criterion the average diversity in the population $avgDiv(P)$ and the maximal number of SGS procedure calls are used.

Algorithm 2 $merMet(P, P_n, pRA)$

```
for each solution S_n in P_n do
    if S_n is obtained from S_k in P then
        if f(S_n) < f(S_k)) then
            add S_n to P
        end if
    end if
    if S_n is obtained from S_k1 and S_k2 in P then
        if f(S_n) < S_k1 or f(S_n) < S_k2 then
            add S_n to P
        end if
    end if
end for
remove from P the worst pRA · |P| solutions
add to P the pRA · |P| of new solutions generated by
genMet
return P
```

Algorithm 3 $genMet(P, pRA, Mg)$

```
generate empty RDD collection
for each mg in Mg do
    generate pRA · |P| · p_mg solutions using mg method
    and add them to RDD
end for
return RDD
```

IV. Computational Experiment

A. Problem instances

To evaluate the effectiveness of the proposed approach 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) [30]. The test set includes 36 instances representing projects consisting from 78 to 200 activities. The file names of the instances are in the form $n\_m\_pr\_st$.def, where n means the number of activities, m the number of resources, pr the number of precedence relations and st the number of skill types. The detailed descriptions and benchmark data analyses can be found in [19], [23].

B. Settings

The computational experiment has been carried out using Intel Core i7 Quad Core CPU 2.6 GHz, 16 GB RAM. The PPMHRL is coded in Scala using Apache Spark environment. In the experiment the following values of parameters have been used:

- Population P of 30 and 50 solutions,
- 5 optimization heuristic algorithms: `LSAm`, `LSAe`, `LSAc`, `PRA`, `EPTA` using the following parameters:
  - $maxiIt_{LSAm} = 20$,
  - $maxIt_{LSAe} = 20$,
  - $maxIt_{LSAc} = 10$,
  - $maxIt_{EPTA} = 10$,
  - $nPart_{EPTA} = 3$,
- $maxSGS = 100000$ - maximal number of SGS procedure calls,
- $minAvgDiv = 0.01$ - minimal average diversity in the population,
- $pRA = 10\%$ - given initial value is decreased when the $avgDiv$ has increased and increased in the opposite case.

TABLE I
PERFORMANCE OF THE PROPOSED PPMHRL SYSTEM IN TERMS OF SCHEDULE DURATION (MAKESPAN).

	ATMAS($\|P\| = 50$)			PPMHRL($\|P\| = 30$)			PPMHRL($\|P\| = 50$)		
Instance	$Best$	AVG	STD	$Best$	AVG	STD	$Best$	AVG	STD
100_5_20_9_D3	392	394	1.67	393	394.8	**1.47**	**388**	**392.4**	2.94
100_5_22_15	**484**	**484.2**	**0.4**	485	485.4	0.49	**484**	484.6	0.49
100_5_46_15	529	530	1.55	529	530.6	1.02	**528**	**529.2**	**0.75**
100_5_48_9	491	491.4	0.49	492	492.2	**0.4**	**490**	**491**	0.63
100_5_64_15	482	483	0.89	482	**482.2**	**0.75**	**481**	482.8	0.98
100_5_64_9	475	475.2	**0.4**	475	475.6	0.8	**474**	**474.8**	0.75
100_10_26_15	237	238.2	**1.17**	**234**	**237.6**	2.73	**234**	238.4	2.58
100_10_27_9_D2	216	225.6	6.47	216	225	**6.42**	**207**	**220.4**	9.97
100_10_47_9	257	257.2	**0.4**	253	255.2	1.72	**252**	**254**	2.28
100_10_48_15	245	246.6	0.8	**244**	**244.2**	**0.4**	**244**	245.4	0.8
100_10_64_9	245	250	3.9	**243**	**246.2**	4.07	244	247.2	**2.32**
100_10_65_15	247	247.4	**0.8**	**244**	246.2	2.04	**244**	**245.6**	1.62
100_20_22_15	136	136	**0**	**130**	**133.2**	2.32	131	**133.2**	1.83
100_20_23_9_D1	174	174.6	**0.8**	**172**	**174**	1.41	174	174.6	**0.8**
100_20_46_15	164	164	**0**	164	164	**0**	**161**	**162.8**	0.98
100_20_47_9	132	133.4	**1.36**	127	132.6	3.01	**124**	**128**	3.41
100_20_65_15	240	240	**0**	240	240	**0**	**220**	**224.8**	3.19
100_20_65_9	140	140.8	**0.75**	140	134	3.35	**124**	**129.2**	5.31
200_10_128_15	464	464	**0**	461	**462.6**	1.02	**460**	**462.6**	1.74
200_10_135_9_D6	710	733.6	**13.4**	642	687.6	38.61	**550**	**603.2**	45.27
200_10_50_15	487	487.8	**0.75**	485	**486.4**	1.74	**484**	487	1.9
200_10_50_9	**488**	**489.6**	1.96	490	491	**1.1**	**488**	490.8	1.72
200_10_84_9	514	514.8	**0.75**	**507**	512.2	3.19	509	**511.2**	2.04
200_10_85_15	476	477.8	1.33	**475**	**477.6**	2.06	477	479	**1.1**
200_20_145_15	245	**246**	**1.1**	245	246.6	1.5	**244**	**246**	1.79
200_20_150_9_D5	**900**	**900**	**0**	910	948.2	20.72	**900**	**900**	**0**
200_20_54_15	268	269.6	**1.02**	263	268.4	2.8	**258**	**262.6**	3.77
200_20_55_9	252	257.6	**3.07**	251	258	4.29	**247**	**256.6**	5.82
200_20_97_15	**336**	**336**	**0**	**336**	**336**	**0**	**336**	**336**	**0**
200_20_97_9	251	251.8	**0.75**	**242**	246.6	4.03	**242**	**245.6**	3.72
200_40_130_9_D4	**513**	**513**	**0**	**513**	**513**	**0**	**513**	**513**	**0**
200_40_133_15	151	156.2	**3.71**	149	154.8	3.87	**135**	**143.8**	8.7
200_40_45_15	164	164	**0**	164	164	**0**	**160**	**162.4**	1.36
200_40_45_9	150	161	9.65	154	160.6	**4.84**	**144**	**152.8**	5.15
200_40_90_9	150	158.6	**6.18**	148	158	7.69	**135**	**148.2**	9.54
200_40_91_15	138	147.4	5.75	138	144.8	**4.66**	**132**	**143.2**	9.37
Average	331.8	334.5	2	328.8	333.6	**3.7**	**322.7**	**327.8**	4

Computations are stopped when the average diversity in the population is less then $minAvgDev$ or the number of SGS procedure calls is grater than $maxSGS$.

C. Results

During the experiment the following characteristics of the computational results have been calculated and recorded: best schedule duration (makespan) ($Best$), average schedule duration (AVG) and standard deviation (STD). Each problem instance has been solved 10 times and the results have been averaged over these solutions.

The computational experiment results for proposed Parallelized Population-based Multi-Heuristic system controlled by Reinforcement Learning strategy (PPMHRL) have been obtained for population size including 30 and 50 individuals are presented in Table I.

The results obtained by PPMHRL are good and promising. The results for the population with 50 individuals are better than for the population with 30 ones. The average $Best$ result is better by an average of 1.9%, the AVG by 1.7%, and simultaneously the STD is slightly lower. Results for both considered population sizes outperform the earlier approaches based on A-Team Multi-Agent Algorithm [12] but in this approach the optimization heuristic algorithms have been modified and one additional optimization algorithm has been used.

Comparison of the obtained results with the results known

TABLE II
COMPARISON WITH THE RESULTS KNOWN FROM THE LITERATURE IN TERMS OF SCHEDULE DURATION (MAKESPAN).

	GRASP			DEGR			GP-HH			PPMHRL($\|P\| = 50$)		
Instance	*Best*	*AVG*	*STD*	*Best*	*AVG*	*STD*	*Best*	*AVG*	*STD*	*Best*	*AVG*	*STD*
100_5_20_9_D3	401	402	**0.6**	392	393.2	0.92	387	387	0	**388**	**392.4**	2.94
100_5_22_15	503	504	1.2	**484**	**484.5**	0.53	**484**	484	0	**484**	484.6	**0.49**
100_5_46_15	552	556	3.9	529	529	**0**	**528**	**528**	**0**	**528**	529.2	0.75
100_5_48_9	510	510	0.3	**490**	490.1	0.32	**490**	**490**	**0**	**490**	491	0.63
100_5_64_15	501	502	1.2	481	483	0.82	481	481	**0**	481	482.8	0.98
100_5_64_9	494	494	0.2	474	474.9	0.32	474	**474**	**0**	474	474.8	0.75
100_10_26_15	251	258	7.1	234	235	1.05	**233**	**233**	**0**	234	238.4	2.58
100_10_27_9_D2	221	223	1.6	215	220.3	2.5	**207**	**207.9**	**0.5**	**207**	220.4	9.97
100_10_47_9	263	264	0.9	255	256.4	0.7	**252**	**252.2**	**0.4**	**252**	254	2.28
100_10_48_15	256	257	1.2	244	245	0.67	**243**	**243.7**	**0.5**	244	245.4	0.8
100_10_64_9	255	257	2.1	243	245.8	1.32	241	**242.2**	**0.6**	244	247.2	2.32
100_10_65_15	256	260	3.8	244	245.3	1.16	**243**	**243.9**	**0.3**	244	245.6	1.62
100_20_22_15	134	137	3.7	130	130.7	0.67	**126**	**126.5**	**0.5**	131	133.2	1.83
100_20_23_9_D1	**172**	172	**0**	**172**	**172**	**0**	**172**	**172**	**0**	174	174.6	0.8
100_20_46_15	170	174	3.9	164	164	**0**	**161**	**161**	**0**	**161**	162.8	0.98
100_20_47_9	133	140	6.8	125	127.5	3.31	**123**	**123**	**0**	124	128	3.41
100_20_65_15	213	213	0.1	240	240	**0**	**205**	**205**	**0**	220	224.8	3.19
100_20_65_9	135	135	0.4	126	129.1	2.73	**123**	**123.8**	0.4	124	129.2	5.31
200_10_128_15	491	496	5.2	461	463.1	0.88	**460**	**460.9**	**0.5**	**460**	462.6	1.74
200_10_135_9_D6	584	617	20.3	608	694.8	67.9	**534**	**534**	**0**	550	603.2	45.27
200_10_50_15	524	528	3.8	487	487.9	0.74	**484**	**484**	**0**	**484**	487	1.9
200_10_50_9	506	508	1.9	485	487.8	1.62	**484**	**484**	**0**	488	490.8	1.72
200_10_84_9	526	527	0.8	507	509.3	2.11	**505**	**505.8**	**0.4**	509	511.2	2.04
200_10_85_15	496	498	1.7	475	478	1.56	**473**	**473.7**	**0.5**	477	479	1.1
200_20_145_15	262	271	8.5	237	238.5	0.71	**236**	237.1	**0.5**	244	246	1.79
200_20_150_9_D5	**900**	913	13.6	**900**	906.9	11.82	**900**	**900**	**0**	**900**	**900**	**0**
200_20_54_15	304	308	3.7	258	261	1.89	258	258.3	0.5	258	262.6	3.77
200_20_55_9	257	258	0.6	249	257.8	10.37	**246**	**246.8**	**0.4**	247	256.6	5.82
200_20_97_15	347	347	**0**	336	**336**	**0**	336	**336**	**0**	336	**336**	**0**
200_20_97_9	253	256	3.8	**241**	247.6	8.93	**241**	**241.4**	**0.5**	242	245.6	3.72
200_40_130_9_D4	**513**	513	**0**	**513**	**513**	**0**	**513**	**513**	**0**	**513**	**513**	**0**
200_40_133_15	163	170	6.5	141	151.4	8.26	**135**	**136.8**	**1**	**135**	143.8	8.7
200_40_45_15	164	164	0.3	164	164	**0**	**159**	**159**	**0**	160	162.4	1.36
200_40_45_9	144	147	3.2	153	182.5	17.83	**137**	**138**	**0.4**	144	152.8	5.15
200_40_90_9	148	153	4.9	148	181.3	22.07	**134**	**135.1**	**0.5**	135	148.2	9.54
200_40_91_15	153	159	5.7	136	144.8	9.44	**130**	**131.6**	**1.1**	132	143.2	9.37
Average	337.6	341.4	3.4	326.1	332.5	5.1	**320.5**	**320.9**	**0.3**	322.7	327.8	4

from the literature are presented in Table II. It can be noticed that the results produced by the proposed approach are comparable with the results from several recently published papers. Among several algorithms proposed for solving MS-RCPSP instances, one seems outstanding and outperforms all others including the proposed one. It is also a population-based algorithm with the search for the best solution enhanced by a hyper-heuristic proposed in [24]. The best makespan value for the GP-HH algorithm is better on average by 0.7% as compared with our approach. It should be noted that the difference in performance between the proposed approach and the GP-HH one gets smaller or even nonexisting as the number of activities increases.

V. CONCLUSION

Results of the computational experiment show that the proposed Parallelized Population-based Multi-Heuristic System control by Reinforcement Learning strategy (PPMHRL) is an efficient and competitive tool for solving MS-RCPSP instances. The obtained results are comparable with solutions presented in the literature.

We believe that there is still room for further improvement of the proposed approach. Future research will focus on finding more effective methods for tuning optimization algorithms parameters. The use of reinforcement learning techniques could be further refined by finding better rules for controlling the number of iterations, population merging, and generating

new individuals. Another performance improvement can be expected from running the solution procedure on a powerful computer cluster that can easily handle a bigger population of individuals and thus profit from the scale and synergy of interactions between optimization agents. It would also be worthwhile to investigate using different types and numbers of optimization algorithms.

REFERENCES

[1] B. F. Almeida, I. Correia and F. Saldanha-da Gama, "Modeling frameworks for the multi-skill resource-constrained project scheduling problem: A theoretical and empirical comparison.", *International Transactions in Operational Research*, vol. 26 (3), 2019, pp. 946–967, https://doi.org/10.1111/itor.12568

[2] O. Bellenguez and E. Néron, "Lower bounds for the multi-skill project scheduling problem with hierarchical levels of skills", *in Proceedings of the international conference on the practice and theory of automated timetabling*, *Lecture Notes in Computer Science*, vol. 3616, Springer, 2004, pp. 229—243, https://doi.org/10.1007/11593577_14

[3] F. Bellifemine, G. Caire and D. Greenwood, "Developing multi-agent systems with JADE." John Wiley & Sons, Chichester, 2007, DOI:10.1002/9780470058411

[4] J. Błażewicz, J. Lenstra and A. Rinnooy, "Scheduling subject to resource constraints: Classification and complexity", *Discrete Applied Mathematics*, vol. 5, 1983, pp. 11–24, https://doi.org/10.1016/0166-218X(83)90012-4

[5] S. Hartmann and D. Briskorn, "A survey of variants and extensions of the resource-constrained project scheduling problem", *European Journal of Operational Research*, vol. 207 (1), 2010, pp. 1–14, https://doi.org/10.1016/j.ejor.2009.11.005

[6] S. Hartmann and D. Briskorn, "An updated survey of variants and extensions of the resource-constrained project scheduling problem", In *European journal of operational research*, vol. 297 (1), 2021, pp. 1–14, https://doi.org/10.1016/j.ejor.2021.05.004

[7] W. Herroelen, E. Demeulemeester, and B. De Reyck, "A classification scheme for project scheduling," *Project scheduling*, 1999, Springer, pp. 1–26, https://doi.org/10.1007/978-1-4615-5533-9_1

[8] A. H. Hosseinian and V. Baradaran, "An evolutionary algorithm based on a hybrid multi-attribute decision making method for the multi-mode multi-skilled resource-constrained project scheduling problem", *Journal of Optimization in Industrial Engineering*, 12 (2), 2019, pp. 155—178, DOI:10.22094/JOIE.2018.556347.1531

[9] P. Jędrzejowicz and E. Ratajczak, "Population Learning Algorithm for Resource-Constrained Project Scheduling," *in Pearson D.W., Steele N.C., Albrecht R.F. (eds) Artificial Neural Nets and Genetic Algorithms*, Springer, Viena, 2003, pp. 223-228, https://doi.org/10.1007/978-3-7091-0646-4_40

[10] P. Jędrzejowicz and P. and E. Ratajczak-Ropel, "Reinforcement Learning Strategies for A-Team Solving the Resource-Constrained Project Scheduling Problem." *Neurocomputing*, vol. 146, 2014, pp. 301–307, doi:10.1016/j.neucom.2014.05.070

[11] P. Jędrzejowicz and E. Ratajczak-Ropel, "Dynamic cooperative interaction strategy for solving RCPSP by a team of agents." *in Nguyen, N.T., Manolopoulos, Y., Iliadis, L., Trawiński, B. (eds) Computational Collective Intelligence*. *Lecture Notes in Artificial Intelligence*, vol. 9875, 2016, pp. 454–463, https://doi.org/10.1007/978-3-319-45243-2_42

[12] P. Jędrzejowicz and E. Ratajczak-Ropel, "A-Team solving multi-skill resource-constrained project scheduling problem", *Procedia Computer Science*, vol. 207, 2022, pp. 3294-–3303 [26th International Conference on Knowledge-Based and Intelligent Information & Engineering Systems (KES 2022)], https://doi.org/10.1016/j.procs.2022.09.388

[13] R. Kolisch, "Efficient Priority Rules for the Resource-Constrained Project Scheduling Problem" *Journal of Operations Management*, vol. 14, 1996, 179–192, https://doi.org/10.1016/0272-6963(95)00032-1

[14] R. Kolisch, "Serial and Parallel Resource-Constrained Project Scheduling Methods Revisited: Theory and Computation" *European Journal of Operational Research*, vol. 90, 1996, pp. 320–333, https://doi.org/10.1016/0377-2217(95)00357-6

[15] R. Kolisch and S. Hartmann, "Experimental Investigation of Heuristics for Resource-Constrained Project Scheduling: An Update", *European Journal of Operational Research*, vol. 174 (1), 2006, pp. 23–37, https://doi.org/10.1016/j.ejor.2005.01.065

[16] J. Lin, L. Zhu, K. Gao, "A genetic programming hyper-heuristic approach for the multi-skill resource constrained project scheduling problem", *Expert Syst. Appl.*, vol. 140, 2020, art. 112915, https://doi.org/10.1016/j.eswa.2019.112915

[17] C. Montoya, O. Bellenguez-Morineau, E. Pinson and D. Rivreau, "Branch-and-price approach for the multi-skill project scheduling problem." *Optimization Letters*, vol. 8 (5), 2014, pp. 1721–1734, https://doi.org/10.1007/s11590-013-0692-8

[18] P. B. Myszkowski, M. E. Skowroński and Ł. Podlodowski, "Novel heuristic solutions for Multi–Skill Resource–Constrained Project Scheduling Problem", *in M. Ganzha, L. Maciaszek, M. Paprzycki (eds) Proceedings of the 2013 Federated Conference on Computer Science and Information Systems FedCSIS*, IEEE, 2013, pp. 159–166, ISBN 978-1-4673-4471-5

[19] P. B. Myszkowski, M. E. Skowroński and K. Sikora, "A new benchmark dataset for Multi-Skill Resource-Constrained Project Scheduling Problem" *in M. Ganzha, L. Maciaszek, M. Paprzycki (eds) Proceedings of the 2015 federated conference on computer science and information systems, ACSIS*, vol. 5, 2015, pp. 129-–138, DOI: 10.15439/2015F273

[20] P. B. Myszkowski, "Hybrid ant colony optimization in solving multi–skill resource–constrained project scheduling problem" *Soft Computing*, vol. 19 (12), 2015, pp. 3599—3619, https://doi.org/10.1007/s00500-014-1455-x

[21] P. B. Myszkowski and J.J. Siemieński, "GRASP applied to Multi-Skill Resource-Constrained Project Scheduling Problem", *in Nguyen, NT., Iliadis, L., Manolopoulos, Y., Trawiński, B. (eds) Computational Collective Intelligence (ICCCI 2016)*, *Lecture Notes in Computer Science*, vol. 9875, 2016, pp. 402—411, https://doi.org/10.1007/978-3-319-45243-2_37

[22] P. B. Myszkowski, Ł.P. Olech, M. Laszczyk and M. E. Skowroński, "Hybrid Differential Evolution and Greedy Algorithm (DEGR) for Solving Multi-Skill Resource-Constrained Project Scheduling Problem", *Applied Soft Computing*, vol. 62, 2018, pp. 1–14, https://doi.org/10.1016/j.asoc.2017.10.014

[23] P. B. Myszkowski, M. Laszczyk, I. Nikulin and M. Skowroński, "iMOPSE: a library for bicriteria optimization in Multi-Skill Resource-Constrained Project Scheduling Problem", *Soft Computing*, vol. 23 (10), pp. 3397–3410, https://doi.org/10.1007/s00500-017-2997-5

[24] E. Néron and D. Baptista, "Heuristics for multi-skill project scheduling problem", *International Symposium on Combinatorial Optimization (CO'2002)*, 2002.

[25] R. Pellerin, N. Perrier, F. Berthaut, "A survey of hybrid metaheuristics for the resource-constrained project scheduling problem", *European Journal of Operational Research*, vol. 280, 2020), pp. 395–416, DOI: 10.1016/j.ejor.2019.01.063

[26] E. Ratajczak-Ropel, "Agent-Based Approach to the Single and Multi-mode Resource-Constrained Project Scheduling. Population-Based Approaches to the Resource-Constrained and Discrete-Continuous Scheduling," *in Janusz Kacprzyk (ed.) Studies in Systems Decision and Control*, vol. 108, Springer International Publishing, 2018, pp. 1–100. doi:10.1007/978-3-319-62893-6.

[27] J. Snauwaert, and M. Vanhoucke, "A classifiction and new benchmark instances for the multi-skilled resource-constrained project scheduling problem," *European Journal of Operational Research*, vol. 307, 2023, pp. 1–19, https://doi.org/10.1016/j.ejor.2022.05.049

[28] Hy. Zheng, L. Wang and Xl. Zheng, "Teaching–learning based optimization algorithm for multi-skill resource constrained project scheduling problem," *Soft Computing*, vol. 21, 2017, pp. 1537–1548. doi.org/10.1007/s00500-015-1866-3

[29] L. Zhu, J. Lin, Y.-Y. Li, and Z. J. Wang, "A decomposition-based multi-objective genetic programming hyper-heuristic approach for the multi-skill resource constrained project scheduling problem," *Knowledge-Based Systems*, vol. 225, 2021, art. 107099, https://doi.org/10.1016/j.knosys.2021.107099

[30] iMOPSE (intelligent Multi Objective Project Scheduling Environment) project homepage, containing description of investigated methods, dataset definition, solution validators, references and best found solutions. http: //imopse.ii.pwr.wroc.pl/