iMOPSE library for benchmarking Multi-Skill Resource-Constrained Project Scheduling Problem

Konrad Gmyrek[1][0009-0000-7206-3674], Michał Antkiewicz[1][0000-0002-6249-4507], Paweł Borys Myszkowski[1][0000-0003-2861-7240], and Jose Luis Calvo-Rolle[2][0000-0002-2333-8405]

[1] Wrocław University of Science and Technology, Faculty of Information and Communication Technology, Wrocław, Poland
{konrad.gmyrek,michal.antkiewicz,pawel.myszkowski}@pwr.edu.pl
[2] University of A Coruña - Department of Industrial Engineering, Coruña, Spain
jlcalvo@udc.es

Abstract. This paper presents an open-source iMOPSE (Intelligent Multi-Objective Problem Solving Environment) library for (meta)heuristic optimization for benchmarking Multi-Skill Resource-Constrained Project Scheduling Problem considered as single-, multi-, and many-objective optimization problems. The library is implemented in C++ and is designed to support researchers, students, and practitioners. The library includes several sets of benchmark instances, implementation of NP-hard problems, and (meta)heuristics, like Genetic Algorithm, Tabu Search, and state-of-the-art multi-objective NSGA-II, SPEA2, or MOEA/D. Additionally, supporting software tools are included, which are helpful in solution validation, visualization, and research automatization. All data and provided code is freely published as open source repository on GitHub.

Keywords: benchmark, scheduling, Multi-Skill Resource-Constrained Project Scheduling Problem.

1 Introduction

The Multi-Skill Resource-Constrained Project Scheduling (MS-RCPSP) is a combinatorial NP-hard scheduling problem related to real-world problems, e.g., the Software Project Scheduling Problem in software development used in Volvo IT company. The tasks that need to be executed are connected in a precedence graph, so the MS-RCPSP problem is overconstrained. Moreover, in MS-RCPSP, the RCPSP problem is extended by resource (human) skills at various levels, introducing additional domain constraints, making the problem more difficult to solve but flexible in management. The MS-RCPSP problem benchmark was originally defined [1][2] and could be considered a single- and many-objective optimization problem. MS-RCPSP problem is commonly used and cited (e.g., surveys [13][14][20]) – according to GoogleScholar nearly 80 scientific papers (years 2015-2024) reference or use MS-RCPSP iMOPSE dataset. Additionally, 13 papers that define the MS-RCPSP problem (and solving methods) are cited 605 times.

Initially, the MS-RCPSP problem was defined as a single-objective optimization problem and presented in [4], where a hybrid of Ant Colony Optimisation (HantCo)

and the greedy algorithm was proposed. The greedy Randomized Adaptive Search Procedure (GRASP) method gained a solution about 15.8% more efficient than HantCo [6]. The above methods solve MS-RCPSP as a single objective using project duration time *Makespan* as the only objective function. Additionally, some of the papers explore the effect of autonomous team role selection in flexible projects and investigate synergies between employees using the iMOPSE dataset [19]. Hybridized Differential Evolution and Greedy (DEGR)[7] can also be used to solve MS-RCPSP, and like all the above methods, is based on a greedy algorithm (Schedule Generator Scheme) to get feasible solutions. Additionally, the extra research presented in [8] presents the influence of how coevolution and solution representation could be effective in solving MS-RCPSP.

The MS-RCPSP can also be solved considering two objectives: project total *Cost* and duration (*Makespan*). In this context, bi-objective optimization in MS-RCPSP should be applied – in [9] results of Non-dominated Sorting Genetic Algorithm (NSGA-II) are presented. However, classical NSGA-II has some limitations, and the new NTGA (Non-dominated Tournament Genetic Algorithm)[10] method has been proposed – NTGA focuses on the diversity of the population, what makes it more effective in solving bi-objective MS-RCPSP. Next, the NTGA2 [11] method has been proposed, which extends the NTGA by GAP selection method and extra mechanism to manage the archive actively. It makes NTGA2 a robust and effective method of multi- and many-objective optimization in solving MS-RCPSP.

To compare the results of NTGA2, NTGA, and classical NSGA-II with other methods, a survey of quality that could be applied directly to MS-RCPSP has been published [12]. A set of complementary measures has been proposed – dedicated and verified in application to MS-RCPSP. In work [11] NTGA2 is investigated in solving MS-RCPSP with 5 objectives and compared to state-of-the-art many-objective methods (e.g. U-NSGA-III or Theta-DEA). Recently, the new *balanced* B-NTGA [18] method has been published – it actively balances the exploitation/exploration in the solution landscape – it dominates the results of other state-of-the-art methods in multi- and many-objective MS-RCPSP.

There are several methods presented in the literature that use iMOPSE MS-RCPSP dataset [2] as a benchmark – they could divided by optimization types: single objective (1 objective), bi–objective (or multi-), and many–objectives (5 objectives). Methods are based on various metaheuristics, like Differential Evolution [7], Ant Colony Optimisation [4], Fruit Fly Optimisation [15] or Teaching-Learning Optimisation [16]. Moreover, hyperheuristics like Genetic Programming Hyper-heuristic [17] are used. Several methods solve MS-RCPSP as a bi-objective problem (e.g., fruit fly MOFOA [21], genetic program. hyper-heuristic MOGP-HH-D [22], NTGA). However, to our best knowledge, only two methods solve MS-RCPSP with 5 objectives: NTGA2 [11] and B-NTGA [18].

The main **contribution** of this paper is (1) to summarise and extend the MS-RCPSP research: all benchmark data [1][2][3] to define a standard set of instances, (2) to provide iMOPSE *open-source* and publish code for state-of-the-art solving MS-RCPSP methods to make method comparison more accessible, and to develop a (3) software tools specialized for MS-RCPSP, like validators or visualizers.

The rest of the article is structured as follows. The definition of MS–RCPSP problem is given in section 2. The MS-RCPSP instances for the investigations are presented in

section 3. Section 4 contains the description of iMOPSE and concludes with a case study showcasing an example of an experiment workflow. The last section (5) concludes the work and highlights potential directions for further research.

2 Multi-Skill Resource-Constrained Project Scheduling Problem

The MS-RCPSP is a combinatorial NP-hard problem within the domain of scheduling problems based on model used in Volvo IT company. The discussed problem comprises two interconnected sub-problems: task sequencing, which involves placing tasks on a timeline, and resource assignments. MS-RCPSP is defined by a list of tasks and resources where each task requires a resource with a specified skill level to be executed. The goal is to find a optimal and a *feasible* schedule PS, meaning a solution that satisfies all **constraints**.

Each resource is linked to a corresponding salary, adhering to the constraint defined in Eq. 1, ensuring that for each resource r, no salary (r_{salary}) value can assume a negative value. Additionally, it dictates that each resource must be associated with a non-empty set of skills, as resources and tasks are linked to specific skill sets.

$$\forall_{r \in R} r_{salary} \geq 0, \forall_{r \in R} S^r \neq \emptyset \tag{1}$$

where S^r is the set of skills possessed by resource $r \in R$.
The duration and finish time of each task cannot be negative (see in Eq. 2.)

$$\forall_{t \in T} F_t \geq 0; \forall_{t \in T} d_t \geq 0 \tag{2}$$

where F_t denotes the finish time, and d_t represents the duration of task t.
Eq. 3 introduces constraints related to task precedence, stating that a task can only start after all its predecessors are completed.

$$\forall_{t \in T, p \in t_p} F_p \leq F_t - d_t \tag{3}$$

where t_p denotes the predecessors of task t.
Eq. 4 addresses the skill requirements in MS-RCPSP, ensuring a resource allocated to a task possesses the requisite skill at an appropriate level.

$$\forall_{t \in T^r} \exists_{s_r \in S^r} h_{s_t} = h_{s_r} \wedge l_{s_t} \leq l_{s_r} \tag{4}$$

where T^r is a set of tasks assigned to a resource r, s_t is the skill required by the task t, S^r is the set of skills possessed by the resource r, h and l are the type and level of the skill respectively.
A constraint ensuring at most one resource is assigned to any task at any given time is presented in Eq. 5.

$$\forall_{r \in R} \forall_{t \in \tau} \sum_{i=1}^{n} U_{i,r}^t \leq 1 \tag{5}$$

where τ is the time domain, n represents the total number of tasks, and $U_{i,r}^t$ is a binary variable, equal to 1 if resource r is assigned to task i at time t.

The final constraint (see Eq. 6) ensures that all tasks must be finished by ensuring that all tasks have a resource assigned at some time slot.

$$\forall_{i \in T} \exists_{t \in \tau, r \in R} U_{i,r}^{t} = 1 \tag{6}$$

where τ and $U_{i,r}^{t}$ are defined as in Eq. 5.

2.1 objectives in MS-RCPSP

The MS-RCPSP can be defined as single- multi- and ultimately as a many-objective optimization framework, accommodating up to five objectives [3].

The optimal schedule is the one with the minimal objective function – for multi-objective optimization, an approximation of Pareto Front is investigated. The feasible schedule satisfies all constraints related to tasks, resources, skills, and precedence relations. Formally, the MS–RCPSP optimization problem can be defined as follows:

$$f : \Omega \rightarrow \mathbb{R}, min(f) \tag{7}$$

where Ω is the feasible schedule space, while the f is the given objective function(s). In many-objective MS–RCPSP there are five defined objectives as follows:

- the project schedule duration (makespan) – f_τ (see Eq. 8),
- schedule's cost – f_C (see Eq.9),
- skill overuse – f_S (see Eq.10),
- and average use of resources – f_R (see Eq.11).
- average cash flow – (f_F) (see Eq.12),

The two most commonly described objectives in literature consist of schedule *Makespan* (or Duration) and total *Cost*. Additional MS-RCPSP objective aims to describe a specific schedule aspect: *Average Cash Flow*, *Skill Overuse*, and the *Average Use of Resources*. The MS-RSPSP optimization objectives are defined below.

The **Makespan** $\mathbf{f}_\tau(\mathbf{PS})$ of the project schedule PS is given as Eq.8.

$$f_\tau(PS) = \max_{t \in T} t_{finish} \tag{8}$$

where T is a set of all tasks, t_{finish} is the finish time of the task t. The **Cost** of the schedule is $\mathbf{f_C}(\mathbf{PS})$ defined as Eq.9.

$$f_C(PS) = \sum_{i=1}^{n} R_i^{salary} * T_i^{duration} \tag{9}$$

where n is the number of all task-resource assignments, R_i^{salary} is the salary of a resource of the i-th assignment, $T_i^{duration}$ is the duration of the task of the i-th assignment.

Skill Overuse aims to minimize the difference between the skill level of a resource and the required skill. **Skill Overuse** $\mathbf{f_S}(\mathbf{PS})$ – see Eq.10) – ensure that the resources

assigned to the task are not overqualified, which could be essential in the practical applications.

$$f_S(PS) = \sum_{i=1}^{n} R_s^l - T_s^l \tag{10}$$

where n is the number of task-resource assignments, R_s^l is the skill level of a resource R, and T_s^l is the skill level required for the task T.

Some resources are assigned to the project - and must receive a salary, even if they are not assigned to any task. The **Average Use of Resources** (see Eq. 11) objective gives the distribution of tasks and ensures the efficient use of resources. It aims to minimize the deviation of the number of task-resource assignments.

$$f_R(PS) = \frac{1}{r} \sum_{i=1}^{r} (R_i^n - R_{avg}^n) \tag{11}$$

where r is the number of resources, R_i^n is the number of tasks assigned to the i-th resource, R_{avg}^n is the expected average number of assignments.

The **Average Cash Flow $\mathbf{f_F}$(PS)** (see Eq. 12) measures the deviation of costs over the entire duration of the project and allows for more effective budget management.

$$f_F(PS) = \frac{1}{f_\tau(PS)} \sum_{t=1}^{f_\tau} (C_t - C_{avg}) \tag{12}$$

where C_t is the cost of the project in a single time slot t, $f_\tau(PS)$ is the makespan of the project, C_{avg} is the average cost of the project in a time unit and can be defined by the Eq.13.

$$C_{avg} = \frac{C}{f_\tau} \tag{13}$$

where C and f_τ are the total $cost$ and $makespan$ of the project respectively.

Although the MS-RCPSP, in its nature is multi-objective, but could also be considered in a simplified version as **single-objective MS-RCPSP**[7], where the evaluation function is formulated as follows:

$$\min f(PS) = w_\tau f_\tau(PS) + (1 - w_\tau) f_c(PS) \tag{14}$$

where: w_τ – weight of duration component, where $f_\tau(PS)$ and $f_C(PS)$ is normalised and $w_\tau \in [0; 1]$.

The above five objectives ($f_\tau, f_C, f_S, f_R, f_F$) are already implemented in iMOPSE library for MS-RCPSP, but there are no limits and this set could be redefined as iMOPSE is opensource.

3 MS-RCPSP instances

The provided dataset emerged during a research cycle and was gradually expanded to accommodate the needs. It consists of 265 unique test instances prepared using

the iMOPSE generator with different parameters to show the influence of constraints (e.g., introducing extremely low and high values for precedence relations or no skill requirements), as well as the task and resource quantity.

All instances are defined using the coherent format and follow the naming convention, encoded as *<task count>_<resource count>_<precedence relation count>_<skill count>_<postfix>*. For example, 200_10_84_9 consists of 200 tasks with 84 precedence relations and 10 resources with up to 9 unique skills. The *<postfix>* at the end is optional and can be used for marking specific variations of the instance. With that said, instance names should not be decoded to access the exact values, as those are nominal values, and slight deviations can be found in the data. Therefore, the exact quantities are included inside the instance definition.

All instances are located within iMOPSE directory *configurations/problems/MSR-CPSP/all* and have been divided into the following groups:

- **Small (6)** - small toy-size instances with 10-15 tasks and 3-9 resources; good for validation and visualization as optimal solutions are easy to find
- **Regular (36)** - regular instances with 100/200 tasks, and 5-40 resources, with varying numbers of skills and relatively small numbers of constraints; instances with postfix 'D' come from the real-world scenarios and might be considered more difficult due to bottlenecks and project milestones; the other instances are generated to imitate the same characteristics
- **RegularGenerated (128)** - regular randomly generated instances with 100/200 tasks, 5-40 resources, 5/10 skills, and varying numbers of constraints; generated to increase the standard instances set providing more combinations to investigate the impact of task, resource, skill, and precedence relation number
- **Dense (7)** - randomly generated instances with 100 tasks, 10-40 resources, 9/15 skills, and a high density of precedence relations
- **NoConstr (8)** - randomly generated instances with 100/200/500/1000 tasks, 20/40 resources, and no constraints (every resource-task connection is valid; no precedence relations)
- **Big (80)** - randomly generated instances with 500/1000 tasks, 10 - 40 resources, 5/10 skills, and varying numbers of constraints

4 iMOPSE library – a general idea

iMOPSE is an advanced, open-source C++ toolkit designed for solving NP-hard problems through a suite of optimization algorithms. Tailored for academic research and practical applications, iMOPSE streamlines the process of addressing complex optimization tasks. The library's modular architecture allows easy extensibility in method and problem implementation (Fig. 1). This section provides an in-depth exploration of iMOPSE capabilities that support research related to the MS-RCPSP problem and methods for solving it.

The iMOPSE library includes several evolutionary-based state-of-the-art multi-objective **methods**, such as NSGA-II [23], Multi-objective Evolutionary Algorithm Based on Decomposition (MOEA/D) [24], Strength Pareto Evolutionary Algorithm

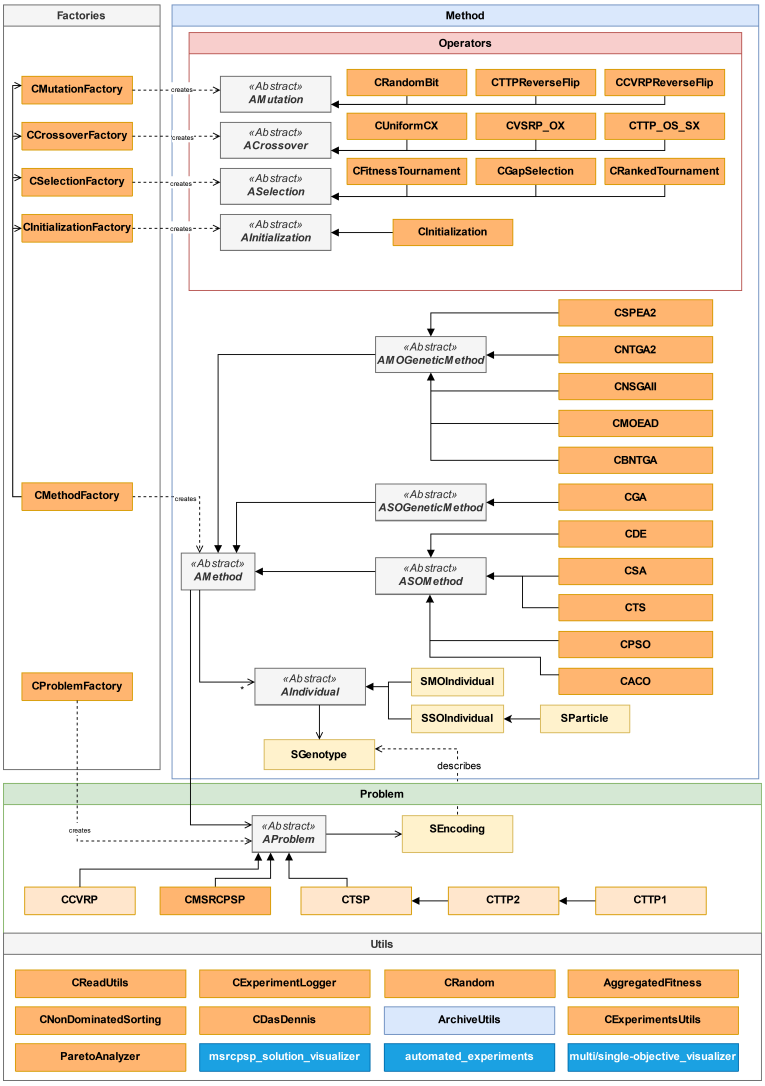

Fig. 1: iMOPSE library – a general schema

(SPEA2) [25], NTGA2 [11] and experimental balanced B-NTGA [18]. Additionally, iMOPSE offers a set of algorithms for single-objective optimization, including Genetic Algorithms (GA), Differential Evolution (DE), Ant Colony Optimization (ACO), Tabu Search (TS), Simulated Annealing (SA), and Particle Swarm Optimization (PSO), therefore providing a set of ready-to-use methods that can be easily extended.

The above methods are known as effective in solving multi-objective NP-hard **problems** with constraints, such as MS-RCPSP [3] and Traveling Thief Problem (TTP). Beyond MS-RCPSP, iMOPSE is capable of handling various classical NP-hard problems, such as the Traveling Salesman Problem (TSP), TTP, and Capacitated Vehicle Routing Problem (cVRP).

Each metaheuristic to search solution space needs a solution representation and defined genetic operators (or neighborhoods). The following two sections describe that.

4.1 representation for MS-RCPSP

For metaheuristics, to enable an effective search in the solution space, a **representation** (solution) should be specialized to a given problem. The iMOPSE library supports three types of representation (permutation, binary, and real-coded), which enable coding NP-hard problems as combinatorial (for example TSP) or priority-based in MS-RCPSP resource assignment (see Fig. 2).

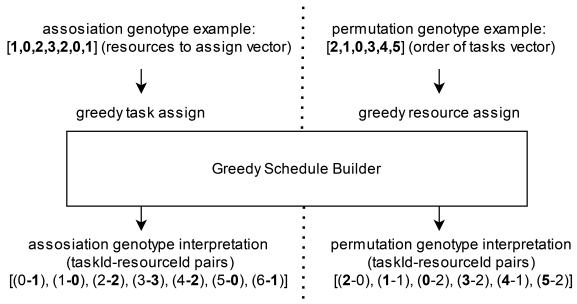

Fig. 2: An example of two types of representation for MS-RCPSP

In the iMOPSE framework, two distinct genotype encodings for the MS-RCPSP are implemented. The first encoding utilizes a vector of task-to-resource associations, wherein each vector element represents a unique resource identifier, and the element's index corresponds to the task identifier. The second encoding adopts a permutation-based approach, wherein the vector describes the sequence of tasks, with each element denoting a task identifier. A dedicated schedule builder uses both encoding methods to construct valid solutions from the genotypes. For the association-based approach, the schedule builder assigns resources by iterating through the genotype vector, assigning the i-th task to the resource indicated by the i-th value in the genotype vector. On the other hand, the permutation-based approach iterates through the sequence of tasks described by the genotype and assigns each task to the resource that will be available in the shortest time. In both scenarios, the greedy schedule builder takes precedence relations into account and automatically adjusts the schedule to ensure validity (see Fig. 2).

Not all solutions can be regarded as *feasible* schedules, as certain constraints might

remain unsatisfied. Each investigated method presented in the following sections, capable of acquiring *feasible* schedule uses a **greedy–based algorithm as Schedule Builder** (i.e.[7][11]) to construct a feasible solution.

4.2 Genetic operators and neighborhoods

Each metaheuristic utilizes **operators** to explore the solution landscape effectively. In iMOPSE, there are several implemented operators: neighborhood operators (for TS or SA), mutations (like *RandomBit*, *ReverseFlip*, *GaussMutation*), or crossovers (e.g., *Ordering Crossover OX*, *Cycle Crossover CX*). Each operator is designed to work with specific encoding types; therefore, users must ensure they are using the correct operator for their chosen encoding method.

To direct metaheuristic in a global search, especially for evolutionary computation, **selection** operators also are needed – *random* (semi-blind, without selection pressure, as reference), classic *tournament* or *gapSelection* [11] for multi-objective optimization. It is worth mentioning that a predefined set of representations, operators, and selections can be easily extended as the iMOPSE library C++ interfaces are given for implementation.

Although the operator architecture in the proposed system already includes specific operations common across various methods, it's entirely feasible to write the entire code for a method within a single class. Nevertheless, we recommend enhancing and building upon the existing operator's architecture or extending it when developing new methods. This approach facilitates more robust and flexible method implementation.

4.3 An additional tools (utils)

The iMOPSE library is equipped with utils that support computation and provide researchers with ready-to-use tools for automation, analysis, and visualization. In this subsection, we describe the most relevant utils tools present in the discussed library (see Fig. 1).

iMOPSE main framework utilizes *ExperimentUtils* and *ExperimentLogger* to support the process of collecting and saving data from experiments, enhancing data management. Additionally, *ArchiveUtils* aids in archive operations for multi-objective methods. For sorting and decomposing optimization problems, Non-dominated sorting is utilized by NSGAII and NTGA2.

External utils – Python scripts The C++ programming language is known as very effective in computation; however, for data analysis and visualization, more useful is *Pyton*. That's why, in the iMOPSE library, several external tools have been added. Our collection of Python scripts is crafted to augment scientific research, each designed for a specific use case and supplemented with descriptive comments for ease of use. *automated_experiments.py* automates the concurrent execution of iMOPSE, offering a robust solution for efficient experimenting. This solution allows researchers to focus more on analysis and less on the operational aspects of their experiments. Furthering our support for scientific analysis of methods, *msrcpsp_solution_visualizer.py* validates the given MS-RCPSP solution against constraints and visualizes it, highlighting broken

constraints and discrepancies (see Fig. 3). This not only aids in improving the understanding of complex optimization solutions but also in refining and enhancing these methods through iterative feedback.

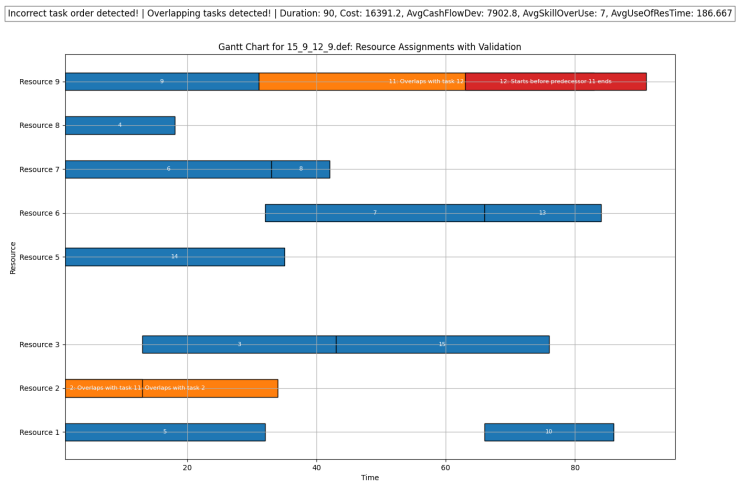

Fig. 3: Example of (invalid) visualized MS-RCPSP solution for instance 15_9_12_9

For researchers working on multi-objective optimization, *multi-objective_visualizer.py* offers visualization of PFA to elucidate the trade-offs between competing objectives. Meanwhile, *single-objective_visualizer.py* provides a graphical overview of the best, worst, and mean fitness values throughout an experiment, enriching the analysis of optimization processes.

Together, these scripts furnish researchers with tools for conducting experiments that are not only more efficient and insightful but also more impactful, thereby enriching the quality and depth of scientific investigations. These scripts serve as rapid, flexible solutions and are planned to be integrated with the main C++ codebase in the future, enhancing the robustness and scalability of the software for broader scientific applications and research.

Pareto Analyzer tool In multi- and many-objective optimization, the output of each method is a set of non-dominated points.A point is dominated if any other point has at least one better (lower, considering the MS-RCPSP) objective value and no worse objective value. All quality measures (QMs) used for multi-objective MS-RCPSP solution are calculated based on the returned set, called Pareto Front Approximation (PFA). The true Pareto Front (TPF) could be defined as a set of all non-dominated solutions and can be considered the best available PFA. However, in practical real-world problems, TPF is usually unknown. *NadirPoint* is a point with the worst possible values for all

objectives. For the MS-RCPSP, the worst value of makespan is the total sum of all tasks' duration - all tasks are serial. The worst cost value is the cost of schedule, where the most expensive resource performs all tasks. Worst skill overuse is achieved by assigning tasks to the resources with the highest required skill level. The worst average use of resources is the maximum makespan multiplied by the number of resources - 1 and divided by the number of resources. The worst value of the average cash flow is the same as the maximum cost.

Commonly used QMs are [12]: *HyperVolume* (HV), *Inverted Generative Distance* (IGD) and *Purity*. All the QMs implemented in the Pareto Analyzer are described below.

HyperVolume ↑ (HV) quantifies the volume of the objective space dominated by a set of solutions. It reflects the spread and coverage of a PF. The higher the hypervolume, the more comprehensive coverage of the objective space. It can be formally defined by Eq. 15.

$$HV(PF) = \Lambda(\bigcup_{s \in PF} \{s' | s \prec s' \prec s^{nadir}\}) \tag{15}$$

where PF is an approximation of PF, s is the point of approximated PF, s^{nadir} is a $NadirPoint$, Λ is a Lebesgue measure, which is the generalization of a volume, $\prec$ is a domination relation.

Inverted Generative Distance ↓ (IGD) captures both convergence and diversity. It is an average distance from each TPF point to the closest point in PF as presented in Eq.16, where d_i is the Euclidean distance for the i-th point. Lower IGD values signify that solutions are closer to the ideal Pareto front. As objectives vary in scale, using absolute values for IGD calculation might favor certain objectives. For that reason, points in PF are normalized beforehand, using minimum and maximum values from the TPF.

$$IGD(PF, TPF) = \frac{\sqrt{\sum_{i=1}^{|TPF|} d_i^2}}{|TPF|} \tag{16}$$

IGD is a relative metric that uses TPF as a reference point. As the real TPF is unknown, it is constructed using results generated by all runs of all compared methods.

Purity ↑ defined as in Eq.17, where ND is the number of solutions (aggregated from all runs) not dominated by the "True Pareto Front approximation" (TPFa), where TPFa is constructed by merging a PFa from each method and removing dominated solutions. $Purity$ calculated for a single method returns the value from 0 to 1 and could be interpreted as the part of TPFa that the given method resulted in. However, the same points (solutions) can be found by different methods. Therefore the sum of $Purity$ for all investigated methods could exceed the value of 1.

$$Purity(PF, TPF) = \frac{|ND(PF, TPF)|}{|TPF|} \tag{17}$$

As each method is evaluated multiple times, the Analyzer repeats the process: merges the results returned by all the methods in the n'-th run, and calculates the $Purity$ per run. At the end, it averages the final $Purity$ value.

A crucial requirement for running the Analyzer is specifying the path to a configuration file as well as the instance name as a mandatory parameters. Configuration file must be prepared in advance and should list paths to the experiment output directory on separate lines. This structured format enables the Analyzer to systematically process and analyze the outcomes for each method listed for a specified instance, ensuring a thorough and organized evaluation process.

iMOPSE supports experiment **reproducibility** through the usage of seed parameters. The first experiment is conducted under a seed provided by the user, and for each next experiment, the seed is achieved by adding one to the initial seed value.

4.4 Case study - performing experiments with iMOPSE

In this section, we introduce an example experiment that users can conduct using iMOPSE, designed to demonstrate its capabilities and help users become acquainted with its operational workflow.

For the case study, we have selected the NSGAII and NTGA2 multi-objective optimization methods to find PFAs for the 200_10_135_9_D6 MS-RCPSP instance. For each method, the experiment will be repeated ten times. Thanks to iMOPSE being outfitted with pre-loaded instances and pre-configured methods, conducting experiments is straightforward. To examine two methods, users can execute the iMOPSE program twice by inputting different parameters (see Fig. 4), or they can utilize the *automated_experiments.py* script, which requires the path to the executable and in this case, two sets of previously mentioned input parameters to run the experiments.

```
../../configurations/methods/NTGA2/NTGA2_ORIGINAL.cfg
MSRCPSP_TA2
../../configurations/problems/MSRCPSP/Regular/200_10_135_9_D6.def
../experiments/NTGA2/200_10_135_9_D6/
10
0
```

```
../../configurations/methods/NSGAII/NSGAII_MSRCPSP.cfg
MSRCPSP_TA2
../../configurations/problems/MSRCPSP/Regular/200_10_135_9_D6.def
../experiments/NSGAII/200_10_135_9_D6/
10
0
```

```
Usage: \imopse.exe <MethodConfigPath> <ProblemName>
<ProblemInstancePath> <OutputDirectory> [ExecutionsCount] [Seed]
```

Fig. 4: iMOPSE input parameters for running NTGA2 and NSGAII

iMOPSE is designed to store the results of each run in a specified output directory. If the directory does not exist, it will automatically create one. Should the output directory already contain data from previous experiments, iMOPSE will halt its operation and notify the user, preventing any loss or accidental overwriting of experiment data due to an incorrect output directory path being provided.

The generated output data can subsequently be analyzed with the aid of additional Python scripts and the *Pareto Analyzer*. Nonetheless, users have the flexibility to employ alternative analysis software or methods according to their preferences.

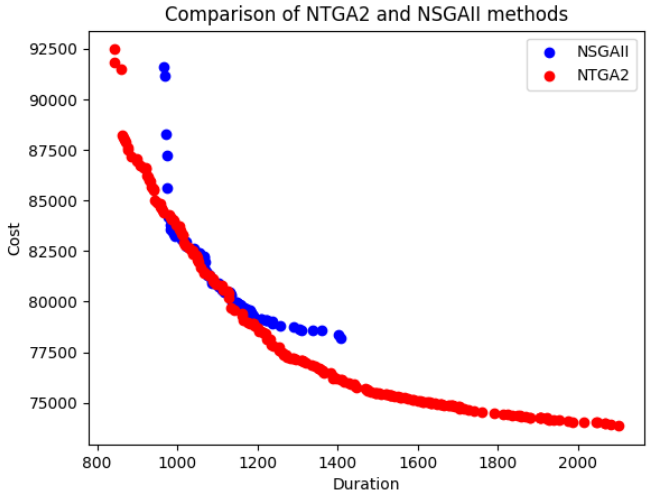

Fig. 5: PFAs comparison for NTGA2 and NSGAII (200_10_135_9_D6 MS-RCPSP)

In the context of our case study, we will utilize the *Pareto Analyzer* to compute metrics and generate a TPF approximation, which will be saved in the same output directory. To run the Pareto Analyzer, the user has to provide a path to the configuration file and the name of the examined instance, in this case, the configuration file contains paths to the output directories of analyzed methods. *Pareto Analyzer* merges results for each method and calculates TPF approximation by taking non-dominated solutions from all methods as reference. QMs acquired by *Pareto Analyzer* in this case study: NSGAII - $HV = 0.56 \pm 0.05$, $IGD = 0.02 \pm 0.003$, $Purity = 0.1$ and for NTGA2 - $HV = 0.76 \pm 0.01$, $IGD = 0.002 \pm 0.0007$, $Purity = 0.9$. The results show that NTGA2 generates approximately 90% of PFA and strongly dominates NSGA-II. Moreover, the PFAs can be visualized and compared by employing the *multi-objective_visualizer.py* script, facilitating a visual comparison of the multi-objective optimization outcomes (see Fig. 5).

5 Summary and future work

This article proposes a new open-source iMOPSE C++ library to support MS-RCPSP researchers, students, and practitioners. The iMOPSE library consists of methods for

solving single- and multi-objective NP-hard combinatorial problems, especially for MS-RCPSP. Additionally, tools are added to validate and visualize MS-RCPSP results. However, the iMOPSE schema is flexible and could be extended easily by adding new methods and/or problems. Thus, the code of iMOPSE is open-source and published on GitHub [5]. The future directions of iMOPSE library development could be connected to support parallel computation. A multi-thread computation and a GPU-based computation of metaheuristics should be considered to speed up computations.

References

1. Myszkowski, P.B., Skowroński M.E., and Sikora K. "A new benchmark dataset for multi-skill resource-constrained project scheduling problem." 2015 Fed.Conf. on Comp. Sci. and Inf. Systems (FedCSIS). IEEE, 2015.
2. Myszkowski P.B., Laszczyk M., Nikulin I., and Skowroński M., "iMOPSE: a library for bicriteria optimization in Multi-Skill Resource-Constrained Project Scheduling Problem", Soft Computing vol.23, (2019), pp.3397–3410.
3. Myszkowski P.B., Laszczyk M., "Investigation of benchmark dataset for many-objective Multi-Skill Resource Constrained Project Scheduling Problem", Applied Soft Computing, Vol 127, (2022), 109253.
4. Myszkowski P.B., Skowroński M.E., Olech L., K Oślizło, "Hybrid ant colony optimization in solving multi-skill resource-constrained project scheduling problem", Soft Comp. 19 (12), 2015, pp.3599-3619.
5. –, http://imopse.ii.pwr.edu.pl – official homepage of iMOPSE project, includes MS-RCPSP resources, GitHub – https://github.com/imopse/iMOPSE
6. Myszkowski P.B., Siemienski J.J., "GRASP Applied to Multi–Skill Resource–Constrained Project Scheduling Problem", Inter. Conf. on Comp. Collective Intelligence, ICCCI 2016, pp.402–411.
7. Myszkowski, P.B., et al. "Hybrid differential evolution and greedy algorithm (DEGR) for solving multi-skill resource-constrained project scheduling problem." Applied Soft Computing 62 (2018): 1-14.
8. Myszkowski P.B., Kalinowski D., and Laszczyk M., "Co-Evolutionary Algorithm solving Multi-Skill Resource-Constrained Project Scheduling Problem", ACSIS 2017, Vol. 11, pages 75–82.
9. Myszkowski P.B., Laszczyk M., Lichodij J., "Efficient selection operators in NSGA-II for solving bi-objective multi-skill resource-constrained project scheduling problem", ACSIS 2017, Vol. 11, pp. 83–86
10. Laszczyk M., and Myszkowski P.B., "Improved selection in evolutionary multi–objective optimization of multi–skill resource–constrained project scheduling problem.", Information Sciences 481 (2019): 412-431.
11. Myszkowski, P.B., and Laszczyk M., "Diversity based selection for many-objective evolutionary optimisation problems with constraints." Inf. Sci. 546 (2021): 665-700.
12. Laszczyk M., and Myszkowski P.B. "Survey of quality measures for multi–objective optimisation. Construction of complementary set of multi-objective quality measures." Swarm and Evolutionary Computation, 2019
13. Hartmann S., Briskorn D., "An updated survey of variants and extensions of the resource-constrained project scheduling problem", European Journal of Operational Research (2022), 297(1), pp.1-14.
14. Verma S., Snasel V., "A Comprehensive Review on NSGA-II for Multi-Objective Combinatorial Optimization Problems", IEEE Access (2021) vol.9., pp. 57757–57791.

15. Zheng Xiaolong, Wang Ling, Zheng Huany, "A knowledge-based fruit fly optimization algorithm for multi-skill resource-constrained project scheduling problem", 2015 34th Chinese Control Conf. (CCC), pp.2615–2620.
16. Huan-yu Zheng, Ling Wang, Xiao-long Zheng, "Teaching–learning-based optimization algorithm for multi–skill resource constrained project scheduling problem", Soft Computing 21, 2017, pp.1537–1548.
17. Jian Lin, Lei Zhu, Kaizhou Gao, "A genetic programming hyper-heuristic approach for the multi-skill resource constrained project scheduling problem", Expert Systems with Applications, vol.140 (2020), 112915.
18. M.Antkiewicz, P.B.Myszkowski, Balancing Pareto Front exploration of Non-dominated Tournament Genetic Algorithm (B-NTGA) in solving multi-objective NP-hard problems with constraints, Information Sciences, (2024), Volume 667.
19. Z.T.Kosztyán, P.Harta, I.Szalkai, The effect of autonomous team role selection in flexible projects, Computers & Industrial Engineering, vol 190, (2024), 110079,
20. J.Snauwaert, M.Vanhoucke, A classification and new benchmark instances for the multi-skilled resource-constrained project scheduling problem, European Journal of Oper. Research, Vol 307 (1), (2023), pp.1-19.
21. Ling Wang, Xiao-long Zheng, A knowledge-guided multi-objective fruit fly optimization algorithm for the multi-skill resource constrained project scheduling problem, Swarm Evol. Comput. 38 (2018) 54–63.
22. Lei Zhua, Jian Lin, Yang-Yuan Li, Zhou-Jing Wang, "A decomposition-based multi-objective genetic programming hyper-heuristic approach for the multi-skill resource constrained project scheduling problem", Knowl.-Based Syst. 225 (2021) 107099.
23. Deb, K. and Pratap, A. and Agarwal, S. and Meyarivan, T., A fast and elitist multiobjective genetic algorithm: NSGA-II, IEEE Trans. on Evol. Comp. (2002).
24. Q.Zhang and H. Li, MOEA/D: A Multiobjective Evolutionary Algorithm Based on Decomposition, IEEE Trans. on Evol. Comp., vol 11(6), pp.712-731, (2007).
25. V.J.Amuso and J.Enslin, The Strength Pareto Evolutionary Algorithm 2 (SPEA2) applied to simultaneous multi-mission waveform design, 2007 Inter. Waveform Diversity and Design Conf., (2007), pp.407-417.