Soft Computing
https://doi.org/10.1007/s00500-017-2997-5

METHODOLOGIES AND APPLICATION

iMOPSE: a library for bicriteria optimization in Multi-Skill Resource-Constrained Project Scheduling Problem

Paweł B. Myszkowski[1] · **Maciej Laszczyk**[1] · **Ivan Nikulin**[1] · **Marek Skowroński**[1]

Abstract
This paper presents a software library as a research and educational tool for Multi-Skill Resource-Constrained Scheduling Problem. The following useful tools have been implemented in Java: instance Generator, solution validator, solution visualizer and example solvers: Greedy algorithm and Genetic Algorithm. All tools are supported by iMOPSE dataset which consists of 36 instances and additional 'small' 6 instances for educational purpose. In the paper, three test studies are described: (1) educational use of 6 'small' instances, (2) optimization of cost or duration of a schedule, and (3) simple bicritieria optimization of cost/duration of a final schedule. All described tools/examples are freely published on iMOPSE homepage.

Keywords Optimization · Scheduling · MS-RCPSP · Software project scheduling problem · Experimental design · Library · Benchmark dataset · Java

1 Introduction

The scheduling problems are the most investigated practical problems that arise in real-life challenges of today's industry, i.e., in manufacturing, chemistry, logistics and many other disciplines. The rare resources and lack of time cause problems that are difficult to solve but also very useful for companies for cost optimization. Multi-Skill Resource-Constrained Project Scheduling Problem (MS-RCPSP) is a specific type of classical RCPSP scheduling problem that is widely described in the literature (Das and Acharyya 2011; Kolisch and Sprecher 1996; Myszkowski et al. 2018). The RCPSP can be defined less formally as a function that assigns jobs to scarce resources to complete the project. Such definition in real world is too general and not useful, because not every resource can be applied to realize a job and additionally jobs are connected by a set of precedence constraints. Such modifications allow for better adaption in applications to manufacturing, production planning, project management, etc. The generalization of (MS-)RCPSP is Project Scheduling Problem (PSP) that is widely described in the literature; however, there is no method that finds an optimal solution and could be applied under every condition. It is known that PSP problems are NP-hard, which means that there is no optimal solution that could be computed in polynomial time (Błazewicz et al. 1983). This is the main reason why researchers try to build methods that give feasible (sub)optimal solution in a time that is acceptable for a user. An agreement is reached in soft computing, where mostly heuristics and metaheuristics (Hartmann and Briskorn 2010; Kolisch and Sprecher 1996) are used.

There are several types of (meta)heuristics that can be effectively applied to MS-RCPSP problem, such as Priority Rules (Skowroński et al. 2013b), Greedy algorithm (Myszkowski and Siemieński 2016; Myszkowski et al. 2015b), Tabu Search (Skowroński et al. 2013a), Simulated Annealing (Yannibelli and Amandi 2013), Genetic Algorithms (GA) (Skowroński and Myszkowski 2013), Bee Colony Optimization (Ziarati et al. 2011), Particle Swarm Optimization (Zhang et al. 2006), Ant Colony Optimization (Myszkowski et al. 2015a), Differential Evolution (Myszkowski et al. 2018), Greedy Randomized Adaptive Search Procedure (GRASP) (Myszkowski and Siemieński 2016), Coevolutionary Algorithms (Myszkowski et al. 2017a) and many others. It is worth mentioning that such methods give solution in acceptable time, but the main disadvantage is the lack of determinism. It means that the same parame-

Communicated by V. Loia.

✉ Paweł B. Myszkowski
pawel.myszkowski@pwr.edu.pl

[1] Collective Intelligence Department, Wrocław University of Science and Technology , Wyb. Wyspiańskiego 27, 50-370 Wrocław, Poland

Published online: 17 January 2018

ter configuration may return different results. However, such methods have many advantages: flexibility, extensibility, and ability to find satisfactory solution. They can be specialized to specific problem to give higher efficiency in real-world applications. The novelty of this paper is mainly connected with complex (software) iMOPSE library presentation, description and publication. Paper presents in details functionality of iMOPSE library and how MS-RCPSP can be solved: methods (like Greedy or GA) and approaches that use criteria (one- or bi-criteria optimization) or weighted sum fitness function. Moreover, to provide didactic potential of iMOPSE library some simple MS-RCPSP instances have been developed.

Proposed iMOPSE library is dedicated to MS-RCPSP problem. The paper describes several MS-RCPSP tools included in the concrete software library, which helps us in development and research process. We want to share our work and developed software platform: iMOPSE (intelligent Multi-Objective Project Scheduling Environment) library. The rest of the paper is organized as follows. Section 2 presents a short summary of existing publications (state of the art). In Sect. 3 the MS-RCPSP problem statement with constraints and requirements is described. The description of proposed iMOPSE library is given in Sect. 4, where its elements are presented: MS-RCPSP instance Generator (see Sect. 4.1), benchmark dataset instances (see Sect. 4.2), validation (see Sect. 4) and visualization tool (see Sect. 4.7). There are presented some simple solvers based on greedy (see Sect. 4.4) and Genetic Algorithm (see Sect. 5.2). Finally, Sect. 5 shows some useful cases where iMOPSE library can be used, as didactic or research tool. Section 6 concludes the article and presents potential directions for future work.

2 Related works

The RCPSP problem is not new and there are some models that support research process. The most popular is PSPLIB library (Kolisch and Sprecher 1996), which is cited by nearly a thousand papers (according to GoogleScholar). It is useful as a dataset to compare solvers efficiency but it uses only classic RCPSP model. It does not support extension of RCPSP that uses resources skills or multi-criteria optimization, e.g., cost and duration of schedule. Moreover, PSPLIB does not contain tools that support research or solution comparison.

Interesting project that supports RCPSP is a didactic RESCON (RESCON 2017), that contains implementation of several (meta)heuristic and other methods, e.g., constructive heuristic, Tabu Search or branch and bound. The RESCON gives a possibility of visualization of the final schedule by several types of charts. The library is written in C++ using Windows platform. However, the project is not supported since 2010.

Very useful dataset of Software Project Scheduling Problem (SPSP) is presented in Luna et al. (2013). There are published on Internet both instances *Generator* and dataset of instances. The SPSP problem is similar to MS-RCPSP, e.g., it is multi-skill problem that can be solved as multi-criteria optimization. There are some differences like tasks-resources assignments, where resource is able to realize more than one task in a given time. More detailed comparison is presented in Myszkowski et al. (2015b).

Each library/dataset is dedicated to a special problem. Each has advantages and disadvantages, but according to our best knowledge there is no library that is connected to MS-RCPSP problem and is supported by several various software tools. Thus, there is a need to find benchmark dataset that is supported by set of tools which can be used for education, e.g., classic Traveling Salesman Problem as NP-problem to be solved by (meta)heuristics or other methods. Also MS-RCPSP can be solved as one of multi-criteria problems that has combinational discrete solution space. Also MS-RCPSP, as practical problem, is potentially very promising direction of scientific research. Such works can be supported by set of test instances and/or instance *Generator* if there is a need to get more instances. Moreover, to verify solution some validation tool can be useful, and finally to "see" the final schedule a visualization tool is needed. In the last few years, we have developed such library and now we want to share it with others.

The proposed iMOPSE library consists of several elements (see the first row in Table 1): applications (instance validator and visualization), programming functionalities (three types of fitness functions and solvers like Greedy or Genetic Algorithm) and benchmark instances (d36 and EDU dataset). However, iMOPSE library is dedicated to MS-RCPSP problem. The base of research is instances (d36) that are solved by several metaheuristics, e.g., GRASP, GA with specialized operators, Coevolutionary Algorithm or Teachinglearning-based optimization (TLBO) algorithm. Presented approaches use different fitness function of MS-RCPSP, e.g., one solves the problem as the minimization of schedule duration, others use weighted fitness functions that connect cost and schedule duration. Other approaches [like MOFA (Wang and Zheng 2018) and NTGA (Myszkowski et al. 2017b)] consider MS-RCPSP as bi-objective problem using Pareto Front.

Presented paper corresponds to other publications summarizing them (see Table 1). Moreover, it presents tools MS-RCPCP like instance Validator and Visualiser. Some functionalities codes (Greedy, GA with Greedy, fitness function usage) and new resources (didactic EDU dataset) first time are presented.

Table 1 Summary of iMOPSE library elements and main references

Generator	MS-RCPSP instances	Solvers	Validation	Fitness type	Visualization
.jar application	d36 (30+ d1-d6)	Greedy	.jar application	Cost/duration	.jar application
	small EDU (6)	GA with Greedy		w_τ weighted bi-criteria	
–	d36: part d1–d6	Heuristics (Skowroński et al. 2013b)	–	Cost/duration	–
–	d36: part d1–d6	Tabu Search (Skowroński et al. 2013a)	–	Cost/duration	–
–	d36: part d1–d6	GA spec. op. (Skowroński and Myszkowski 2013)	–	Cost/duration	–
–	d36	Greedy (Myszkowski et al. 2015b)	–	Cost/duration	–
–	d36	GRASP (Myszkowski and Siemieński 2016)	–	Cost/duration	–
–	d36	HantCo (Myszkowski et al. 2015a)	–	Duration/cost	–
–	d36	HantCo (Myszkowski et al. 2018)	–	w_τ Weighted	–
–	d36	Coev.Algorithm (Myszkowski et al. 2017a)	–	Duration	–
–	d36	DEGR (Myszkowski et al. 2018)	–	Duration/cost	–
–	d36	DEGR (Myszkowski et al. 2017b)	–	w_τ weighted	–
–	d36	NTGA (Myszkowski et al. 2017b)	–	Bi-criteria	–
–	d36	TLBO (Zheng et al. 2017)	–	Duration/cost	–
–	d36	MOFA (Wang and Zheng 2018)	–	Bi-criteria	–

3 Formulation of MS-RCPSP

The goal of MS-RCPSP is to find a schedule that is both feasible and optimal. Optimal schedule is the one that has the smallest fitness function value f, while feasible schedule is the one that satisfies all constraints on resources, skills and task dependencies. We can formally state the problem as:

$$f : \Omega \rightarrow \mathbb{R}, \min(f) \tag{1}$$

where Ω is the feasible solution space and f is the fitness function. Two principal fitness functions for MS-RCPSP are f_C—for cost-based optimization, and f_τ—for time-based optimization.

Feasible schedule (PH) comprises the set of tasks $T = 1 \ldots n$, the set of resources $R = 1 \ldots m$, and the set U defining assignments of tasks to the resources. We can formally describe PH as:

$$PH := (T, R, U) \tag{2}$$

In order to define the tasks set T, first the set of skills Q ($|Q| = k$) has to be introduced. The skills set Q can be defined as follows:

$$Q := \left\{ Q_i : \forall_{i \in \mathbb{N}, i<k}\ q_i := (h_{q_i}, l_{q_i}) \right\} \tag{3}$$

given:

$$h_{q_i}, l_{q_i} \in \mathbb{N} \tag{4}$$

and h_q is the type of skill q, and l_q is the level of skill q. Using the above definition of the skills set Q, we can now define tasks set T as follows:

$$T := \left\{ t_i : \forall_{i \in \mathbb{N}, i<n}\ t_i := (d_{t_i}, s_{t_i}, f_{t_i}, q_{t_i}, P_{t_i}) \right\} \tag{5}$$

given:

$$d_{t_i}, s_{t_i}, f_{t_i} \in \mathbb{N},\ f_{t_i} = s_{t_i} + d_{t_i}, q_{t_i} \in Q,\ P_{t_i} \subseteq T \setminus \{t\} \tag{6}$$

and d_t is the duration of task t, s_t is the start time of task t, f_t is the finish time of task t, q_t is the required skill in order to complete task t, P_t is the set of predecessors of task t. Given the task t, we can define the set of its predecessors as follows:

$$\forall_{p \in P_t}\ f_p \leq s_t \tag{7}$$

The interpretation of (7) is such that the task t cannot be started until all of its predecessors in P_t are finished. Next, we can define the resources set R as follows:

$$R := \left\{ r_i : \forall_{r \in \mathbb{N}, i<m}\ r_i := (c_{r_i}, Q_{r_i}) \right\} \tag{8}$$

given:

$$c_{r_i} \in \mathbb{R}^+,\ Q_{r_i} \subseteq Q \tag{9}$$

and c_{r_i} is the salary of resource r_i, Q_{r_i} is the set of skills of resource r_i. Each resource must have at least one skill, which can be formally described by the following constraint:

$$\forall_{r \in R}\ Q_{r_i} \neq \emptyset \tag{10}$$

Given the resource r_i, we can define the subset $T_r \subseteq T$ of feasible tasks that contains all the tasks that can be completed by resource r_i. The formal definition is as follows:

$$T_r := \left\{ t \in T : \exists_{q^* \in Q_r}\ h_{q^*} = h_{q_t} \wedge l_{q^*} \geq l_{q_t} \right\} \tag{11}$$

We define schedule execution time as the maximum finish time for all tasks in T, namely:

$$f_\tau(PH) := \max_{t \in T} f_t \tag{12}$$

Given the discrete nature of time in MS-RCPSP, we introduce the notion of time-slot τ to be the quantum of time that belongs to the domain $\tau \in \mathbb{N}$. Therefore, we further introduce the notation $U^\tau_{t,r} \in \{0, 1\}$, that is equal to 1 if the task t is being executed by resource r at the time τ, and is equal to 0 otherwise. Using this notation, we can define the constraint on resources that prevents resources from executing several tasks simultaneously:

$$\forall_{r \in R} \forall_{\tau \in \mathbb{N}} \sum_{t \in T} U^\tau_{t,r} \leq 1 \tag{13}$$

Furthermore, we can define the task-resource assignment function:

$$\alpha : T \rightarrow R \tag{14}$$

We assume that $\alpha(t) = r$ means that resource r starts the execution of task t without preemption at time s_t and finishes it at time f_t. The formal constraint can be stated as follows:

$$\alpha(t) = r \Rightarrow \forall_{s_t \leq \tau \leq f_t}\ U^\tau_{t,r} = 1 \wedge \forall_{\tau \notin [s_t, f_t]}\ U^\tau_{t,r} = 0 \tag{15}$$

Using the assignment function, we can now define the cost-based f_C fitness function as follows:

$$f_C(PH) := \sum_{t \in T} d_t \cdot c_{\alpha(t)} \tag{16}$$

MS-RCPSP is bi-criteria optimization problem, where minimization criteria are cost of the schedule, as well as its duration. Solving method can consider problem as multi-criteria, e.g., using dominance relation and Pareto Front measures.

However, MS-RCPSP problem can be also defined with weighted fitness function, that connects f_τ and f_c (before that each value should be standardized) as follows:

$$\min\ f(PH) := w_\tau f_\tau(PH) + (1 - w_\tau) f_c(PH) \tag{17}$$

Such approach allows to use each (meta)heuristics method as a solver. Moreover, definition of w_τ allows to prioritize schedule cost and duration factor. For example $w_\tau = 1.0$ means duration optimization, value $w_\tau = 0.0$ means cost optimization. It is worth to notice that values between smoothly explore the whole problem landscape. The example of such approach is given in Sect. 5.3.

4 iMOPSE library

The proposed iMOPSE library consists of several applications: instances *Generator* (see Sect. 4.1), two datasets of predefined instances (Sect. 4.2), solution validator (sec 4.6) and solution visualizer (see Sect. 4.7). These tools are developed for all researchers to make possible not only to generate new instances, but also to check if provided solution is valid and optimal. For educational purpose, we have developed Java source codes of Greedy Algorithm (see Sect. 4.4) and Genetic Algorithm (see Sect. 4.5) to show how easy it is to extend it.

All presented tools (and documentations) are freely published as Java *jar* in iMOPSE project homepage.[1] Moreover, to keep operating system independence we decided to use Java programming language.

4.1 Instance generator

Such application has been created to generate new MS-RCPSP instances. The example of Generator application usage is provided in Fig. 1. To build new instance several task's attributes are needed: number of tasks, duration (minimal and maximal) and number of (precedence) relations. Next, resource configuration should be given: number of resources, standard rate (minimal and maximal), overtime rate (minimal and maximal) and number of skills (minimal and maximal).

The last step is to set output filename and press Generate button. After generation procedure, the instance is created and if option "Assign resources" is applied in *.def file each task is connected to resources. Moreover, for given instance, difficulty measures are estimated. More about difficulty measures and *Generator* algorithm can be found in Myszkowski et al. (2015b).

4.2 Predefined instances: small and benchmark iMOPSE

Using *Generator* application, two datasets have been generated: simple for *EDU*(cational) usage that consist of 6 instances and benchmark dataset *d36* of 36 instances for

[1] http://imopse.ii.pwr.wroc.pl/

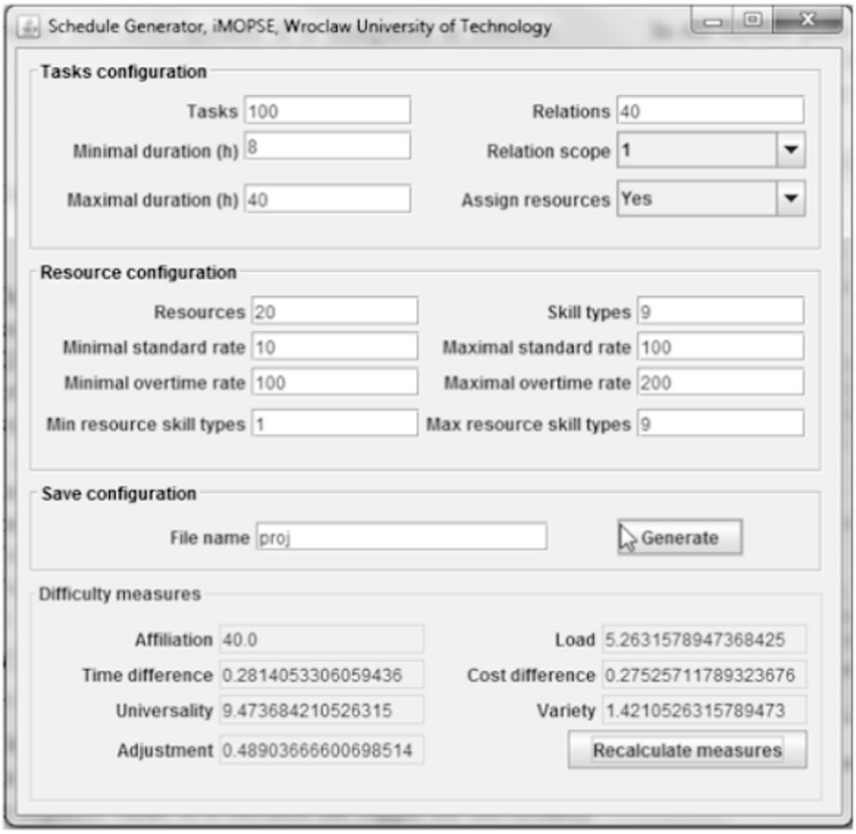


Fig. 1 Example of iMOPSE generator usage

Table 2 Summary of iMOPSE EDU dataset

Instance	Tasks	Resources	Relations	Skills
10_3_5_3	10	3	5	3
10_5_8_5	10	5	8	5
10_7_10_7	10	7	10	7
15_3_5_3	15	3	5	3
15_6_10_6	15	6	10	6
15_9_12_9	15	9	12	9

research experiments. The *EDU* dataset consists of only a few tasks (10 or 15, see Table 2) and is useful for educational usage. It can be solved to prove if examined method works properly, and solution can be easily analyzed by students, validated and visualized by tools.

However, in research the simple *EDU* dataset is no challenge. Thus, we provided and published dataset of 36 instances (Myszkowski et al. 2015b) (see Table 3) to compare results of given methods (e.g., Myszkowski et al. 2015a; Myszkowski and Siemieński 2016). It differs not only in number of tasks, but also in number of resources, relations and used skills.

The predefined datasets (EDU and benchmark d36) can be used for various applications but other applicable instances can be created easily using Generator application.

4.3 iMOPSE - a library project

To simplify the usage of iMOPSE library, the generalized class diagram is shown in Fig. 2. The library is split into several packages according to the functionality of classes, which in their turn may contain further subpackages. The principal package is **scheduling** package, which provides the interface to the MS-RCPSP scheduling problem. Classes **Schedule**, **Task**, **Skill**, and **Resource** correspond to respective entities in the definition of MS-RCPSP. Instance of **Schedule** contains instances of **Task** and **Resource**, which can be manipulated through the public interface of **Schedule**. **BaseIndividual** is a wrapper class for **Schedule**, that allows to evaluate schedules using **BaseEvaluator** instances and compare it to other schedules. **BaseIntIndividual** is an extension of **BaseIndividual** that stores additional information about individual's genes using problem representation coded by integer values. The **schedule_builders** subpackage contains abstract **ScheduleBuilder** class that aims to build a feasible schedule from the arbitrary schedule, along with its two proposed implementations: **ForwardScheduleBuilder** and **BackwardScheduleBuilder**.

The **evaluation** package comprises the classes that allow to evaluate schedules and allow to compare schedules based on different factors. **BaseEvaluator** is the base abstract class for evaluation that has three concrete implementations:

Table 3 Summary of iMOPSE benchmark d36 dataset (Myszkowski et al. 2015b)

Instance	Tasks	Resources	Relations	Skills
100_20_23_9_D1	100	20	23	9
100_20_22_15	100	20	22	15
100_20_47_9	100	20	47	9
100_20_46_15	100	20	46	15
100_20_65_9	100	20	65	9
100_20_65_15	100	20	65	15
100_10_27_9_D2	100	10	27	9
100_10_26_15	100	10	26	15
100_10_47_9	100	10	47	9
100_10_48_15	100	10	48	15
100_10_64_9	100	10	64	9
100_10_65_15	100	10	65	15
100_5_20_9_D3	100	5	20	9
100_5_20_15	100	5	22	15
100_5_48_9	100	5	48	9
100_5_48_15	100	5	46	15
100_5_64_9	100	5	64	9
100_5_64_15	100	5	64	15
200_40_45_9	200	40	45	9
200_40_45_15	200	40	45	15
200_40_90_9	200	40	90	9
200_40_91_9	200	40	91	15
200_40_130_9_D4	200	40	130	9
200_40_144_15	200	40	133	15
200_20_55_9	200	20	55	9
200_20_54_15	200	20	54	15
200_20_97_9	200	20	97	9
200_20_97_15	200	20	97	15
200_20_150_9_D5	200	20	150	9
200_20_145_15	200	20	145	15
200_10_50_9	200	10	50	9
200_10_50_15	200	10	50	15
200_10_84_9	200	10	84	9
200_10_85_15	200	10	85	15
200_10_135_9_D6	200	10	135	9
200_10_128_15	200	10	128	15

CostEvaluator, **DurationEvaluator**, and **WeightedEvaluator**. First two implementations use cost and duration as fitness factors, respectively, while the last one combines cost and duration factors using a weighted linear combination of both factors.

The **validation** package allows to check schedule feasibility and diagnose the potential errors. **AssignmentValidator**, **ConflictValidator**, **RelationValidator**, and **SkillValidator** check the existence of assignment, resource, relation, and skill conflicts, respectively. **CompleteValidator** in its turn launches all other validators, thus providing a simple interface to check overall schedule feasibility.

The **evolutionary_algorithms** package provides basic implementation of evolutionary algorithms, that operates on **Schedule** and uses **BaseEvaluator** and **ScheduleBuilder** to evaluate and manipulate its elements. The subpackage named **genetic_algorithm** contains the concrete and more specialized implementation of genetic algorithm. The subpackage is split even further into several subpackages that are not shown on the class diagram and concentrate on the different elements of genetic algorithm. Base **EvolutionaryAlgorithm** class provides the possibility to implement other evolutionary algorithms by subclassing it.

Visualization package encapsulates the visualization functionality of the library. **Visualizer** serves as the main interface to interact with schedules. It uses **CriticalPathBuilder** to highlight the tasks that belong to the critical path of the project. It also uses instance of **Renderer** to generate the HTML representation of the schedule based on the preprocessed data. **TimeslotTable** is the essential implementation of **Renderer** interface that uses **Palletizer** to help generate new colors for the tasks. As in **EvolutionaryAlgorithm** package, it is possible to implement custom visualization renderers by implementing **Renderer** interface.

Finally, **io** package consists of single **MSRCPSPIO** class that allows to read project definitions from files, as well as read and write solutions for existing projects.

4.4 Solver: Greedy

The primary purpose of the library is to support development of MS-RCPSP solving methods. In scientific context, it is often necessary to have a baseline approach, to which comparisons can be easily made. In educational context, a simple approach has been developed. It illustrates the basic mechanisms of the MS-RCPSP. Greedy approach is a perfect fit for iMOPSE simplicity presentation and educational purpose.

Greedy approach to MS-RCPSP is essentially twofold. Either task-resource assignment is given and the aim is to find optimal start times of each task, or times are given and task-resource assignments are the subject of the search. Below the former is presented.

Algorithm 1 describes Greedy Algorithm applied to MS-RCPSP that optimize schedule duration (using $w_\tau = 1.0$ value). The idea behind it is to first process tasks that have successors and then, process the rest. It allows to manage tasks with the most dependencies as soon as possible and then fit remaining work around them. Lines from 2 to 6 process tasks with successors and lines from 7 to 11 process tasks without successor, in a similar manner. In lines 4 and 9 a variable is set, which represents the finish time of the

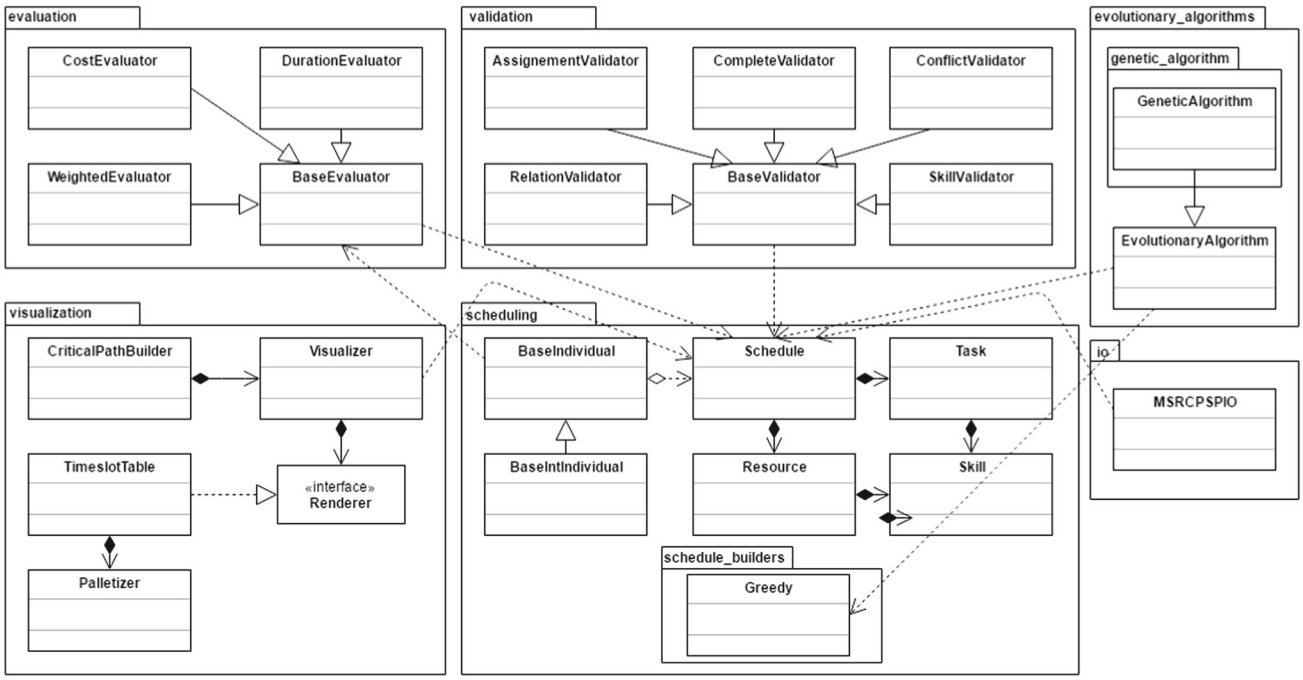


Fig. 2 The general class diagram of iMOPSE library

Algorithm 1

```
procedure GREEDY ()
  for task ← 1, numberOfTasks do
    if task has successors then
      maxPred ← end time of last predecessor
      minRes ← min time of available resource
      task.start ← max(maxPred, minRes)
    end if
  end for
  for task ← 1, numberOfTasks do
    if task has no successors then
      maxPred ← end time of last predecessor
      minRes ← min time of available resource
      task.start ← max(maxPred, minRes)
    end if
  end for
   return Schedule
end procedure
```

last predecessor. However, it does not guarantee a resource would be available at that time. Hence lines 5 and 10 look for the earliest time a resource can start working on the given task. Max of those two variables is the earliest time the task can get started and so it is assigned in lines 6 and 11.

The iMOPSE library allows to run Greedy 1 in a very simple way. Listing 1 shows a function written in Java which does exactly that. Note, that all functionality related to reading the definition and handling the problem is included in the library.

Listing 1 Greedy MS-RCPSP solver

```
private static List<BaseIntIndividual>
    run(String[] args) {
  MSRCPSPIO reader = new MSRCPSPIO();
  Schedule schedule =
      reader.readDefinition(definitionFile);
  if (null == schedule) {
    LOGGER.log(Level.WARNING, "Could not
        read the Definition " +
        definitionFile);
    return null;
  }

  int[] upperBounds =
      schedule.getUpperBounds(
                schedule.getTasks().length);
  BaseEvaluator evaluator = new
      WeightedEvaluator(schedule, 1.0);
  Random generator = new Random(seed);
  BaseIntIndividual resultIndividual = new
      BaseIntIndividual();
  List<Integer> genes = randomGenes();
  resultIndividual.setGenes(genes);
  Schedule resultSchedule =
      resultIndividual.getSchedule();

  for (int i = 0; i <
      resultIndividual.getGenes().length;
      ++i) {
    resultSchedule.getTasks()[i]
     .setResourceId(resultIndividual.
       getGenes()[i]+1);
  }
  resultSchedule.buildTaskResource
     Assignments();
```

```
25
    Greedy scheduleBuilder = new
        Greedy(schedule.getSuccesors());
    scheduleBuilder.buildTimestamps
      (resultSchedule);

30  BaseValidator validator = new
        CompleteValidator();

    System.out.println(validator.validate
       (resultSchedule));
    resultIndividual.setDurationAndCost();
35  System.out.println("Duration: " +
        resultIndividual.getDuration());
    System.out.println("Cost: " +
        resultIndividual.getCost());

    Arrays.stream(resultSchedule.getTasks()).
      forEach((t) -> t.setStart(t.getStart() +
          1));
40  Visualizer visualizer = new Visualizer(new
        TimeslotTable());
    String html =
        visualizer.Visualize(resultSchedule,
        true);
    try {
      FileWriter writer = new
          FileWriter(visualizationFile);
      writer.write(html);
45    writer.close();
    } catch (IOException e) {
      e.printStackTrace();
    }

50  try {
      reader.write(resultSchedule, writeFile);
    } catch (IOException e) {
      System.out.print("Writing to a file
          failed");
    }
55
    return resultSchedule;
  }
```

First lines in 1 use **MSRCPSPIO** object to read the definition file. In line 9, an **upperBounds** variable is created, which consists of upper bounds for each gene. Evaluator in line 11 takes a weight value as a parameter to determine whether to use cost or duration optimization value 1.0 means duration optimization ($w_\tau = 1.0$). Then, a **BaseIndividual** object is used, which wraps a **Schedule** and stores approach specific values to it, such as evaluation value or genes. Next, *randomGenes()* return random vector to set the initial schedule. In a loop that begins in line 18 tasks have their resources assigned based on the genes values. In line 21 schedule ensures that only viable resources are used. Finally in line 23 a **ScheduleBuilder** is used to perform Algorithm 1. Following lines show how to validate a **Schedule** and extract duration, naturally a cost can be taken in a similar way. Normalized values for both cost and duration are also stored within the individual. Starting with the line 35 a visualizer is used to create a visualization file, which is then saved within a try clause in line 36. Line 44 begins another try clause which aims to save a solution file.

4.5 Solver: Greedy guided by Genetic Algorithm

In the iMOPSE library, a Greedy Algorithm guided by Genetic Algorithm has been implemented to solve MS-RCPSP. It has been designed to allow for interchangeability of its genetic operators. It lets user write custom operator and put it in place with minimum effort. The idea behind this architecture was to accelerate research and let student to focus on the mechanism instead of implementation. Genetic Algorithm uses a combination of simple parameters and genetic operators. Evaluation rate determines whether duration or cost of the schedule is the subject of optimization. Population size determines the number of individuals in the population. Higher values of this parameter allow better coverage of optimization space but also increase the time required to run the algorithm. There is also a generation limit which is used as a default stopping criteria. Complex parameters used here are the initial population generation method and genetic operators—selection, crossover and mutation types. Mutation requires an additional parameter, which represents probability of mutating each gene. Example implementation of each operator has been provided.

A vector representation of the genome has been implemented. For each task it assigns an id of a resource. The following can be exemplary individual: (5,2,1,2), which means that the first task has the fifth resource assigned. The second task has the second resource assigned and so on. This approach skips setting starting times of each task. For this a greedy algorithm could be used. It iterates over all tasks and assigns the earliest possible time for them, considering the time at which its assigned resource finishes its previous work and also finish time of all its predecessors. The Greedy algorithm functionality is provided in the library.

Genetic algorithm usually needs two genetic operators to work: crossover and mutation. Crossover is performed by swapping genetic material between individuals. The simplest way to achieve it is classic one-point crossover (OX). For example, having two parent individuals: P_1:=(1, 2, 3, 4, 5) and P_2:=(6, 7, 8, 9, 10) and setting a slice point at the third position, we get two offspring: C_1:=(1, 2, 3, |9, 10) and C_2:=(6, 7, 8, |4, 5).

In case of mutation, the operator works on a single individual and, in the simplest approach, randomizes the value of each gene with given probability. For example, having individual P:= (1,2,3,9,10) and randomizing its third gene (values 3 changes to 10), the individual mutates into individual P':= (1,2,10,9,10).

The Pseudocode 3 describes a Greedy-based approach guided by Genetic Algorithm. It begins with initializing a

```
1: procedure GeneticAlgorithm ()
2:     population ← initialPopulation()
3:     population ← evaluation()
4:     while stopping criteria not met do
5:         newPopulation := ∅
6:         for individual ← population do
7:             parents ← selection()
8:             offspring ← crossover(parents)
9:             offspring ← mutation(offspring)
10:            buildScheduleGreedy(offspring)
11:            newPopulation.add(offspring)
12:        end for
13:        population ← newPopulation
14:        population ← evaluation()
15:        bestIndividual ← population.getBest()
16:    end while
       return bestIndividual
17: end procedure
```

Fig. 3 Greedy guided by Genetic Algorithm solves MS-RCPSP

new population in line 2 and evaluates first solutions. Then, a loop in line 4 runs until the stopping criteria are met. Each iteration begins with creating an empty population, which will hold future population in line 5 and then it is completed by processing old population. Two parents are selected in line 7, then by using crossover, two offspring individuals are created in line 8. Next, mutation is performed in line 9 to get diversity in population. Lastly, Greedy schedule builder is used to transform genes into task-resource assignments and evaluate the schedule in line 10. Created individual(s) is added into the new population (line 11). When newly created populations size equals to the size of last population, the latter is replaced with the new one.

To run this approach in the iMOPSE library a definition file must be read and all parameters must be defined. The functionality for the former is already provided and for the simple parameters the user has to simply provide the values. For each of the complex parameters, enumerator is provided with each of the implemented methods. The user has to either choose one of them or create his own implementation. To do the latter, an enumerator has to be extended with the name of the new method along with the setter inside **GeneticAlgorithm** class. The method itself has to be provided inside a class, which extends the base class for corresponding operator (i.e., **BaseMutation** as mutation procedure).

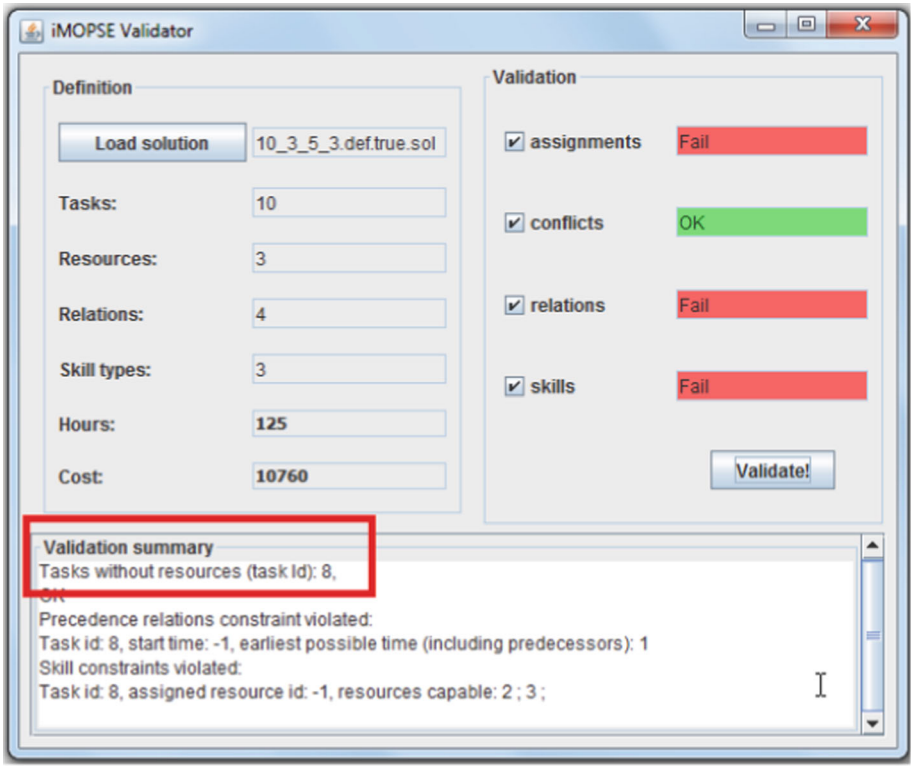
Listing 2 Greedy guided by Genetic Algorithm solves MS-RCPSP

```
1  private static List<BaseIntIndividual> run()
       {
     MSRCPSPIO reader = new MSRCPSPIO();
     Schedule schedule =
         reader.readDefinition(definitionFile);
     Schedule resultSchedule;
5    if (null == schedule) {
       LOGGER.log(Level.WARNING, "Could not
           read the definition file " +
           definitionFile);
     return null;
   }

10 int populationSize = 100;
   int generationLimit = 50;
   double tauValue = 1.0;
   double mutationProbability = 0.1;
   double crossoverProbability = 0.8;

15 InitialPopulationType
       initialPopulationType =
       InitialPopulationType.RANDOM;
   SelectionType selectionType =
       SelectionType.ROULETTE_WHEEL;
   CrossoverType crossoverType =
       CrossoverType.SINGLE_POINT;
   MutationType mutationType =
       MutationType.RANDOM;
20 ScheduleBuilderType scheduleBuilderType =
       ScheduleBuilderType.
       FORWARD_SCHEDULE_BUILDER;
   BaseEvaluator evaluator = new
       WeightedEvaluator(schedule, tauValue);

   GeneticAlgorithm geneticAlgorithm = new
       GeneticAlgorithm(
       schedule, populationSize,
           generationLimit,
           initialPopulationType,
           selectionType,
       crossoverType, mutationType,
           evaluator, scheduleBuilderType,
           mutationProbability,
           crossoverProbability);

   List<BaseIntIndividual> resultIndividuals
       = geneticAlgorithm.schedule();

30 return resultIndividuals;
 }
```

Listing 2 shows how to run Greedy-based approach guided by Genetic Algorithm. Lines 2-9 read the definition file. Later on the parameters are defined. First in lines 10–14 the simple parameters are defined. In lines 15–21 complex parameters are chosen by providing appropriate value from the enumerator. In line 22 **GeneticAlgorithm** is created with all parameters. Ultimately in line 23 the genetic algorithm is run and result individuals are stored in a list. Results from the base approach consist of only one individual, but the list allows for future multi-criteria extension.

4.6 Solution validator

In MS-RCPSP there are several constraints (described in Sect. 3) that should be satisfied: every task must have assigned resource, assigned resource must have skills required by the task and each resource cannot realize more than



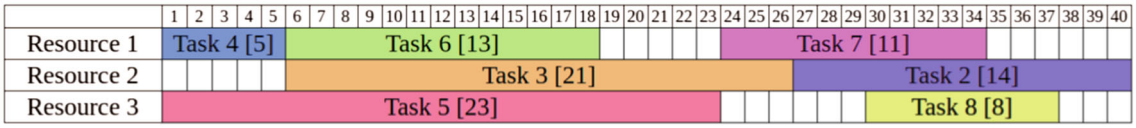


Fig. 4 Example of iMOPSE *validator* application usage

Fig. 5 Sample schedule visualization

one task at a given time. Moreover, realization of a task must be valid according to precedence relations. Duration and cost of the schedule can be calculated only if the solution is valid and represents a feasible schedule. In Fig. 4 an example of an invalid schedule is presented with broken constraints. The *validator* application reports where/which constraints are not satisfied. It potentially helps the developer/researcher to fix the examined solving method.

According to our didactic experience, the reference *validator* tool is especially useful for students to examine if provided solution is valid.

4.7 Solution visualization

In the iMOPSE library, the visualization of MS-RCPSP solutions has been implemented to improve the analysis of optimization approaches. The visualization focuses on the time-based optimization and does not take cost factor into consideration. The example of a visualization is shown in Fig. 5. The visualization is represented by the table, where rows correspond to resources and columns correspond to time-slots. Rows contain merged blocks of adjacent cells that correspond to separate tasks executed by respective resource at appropriate time-slots. For example, the block in the middle represents the task number 3 which is assigned to resource number 2. The tasks execution begins at time-slot 6 and finishes at time-slot 26. The number inside the brackets represents the task duration. The width of the table, therefore, represents the total time of schedule execution.

One of the main properties of developed visualization is that it provides good estimate of how good the solution is. Taking that into consideration, the additional feature of the visualization has been implemented. The feature allows to highlight the tasks that are on the critical path (or one of the paths, in case there are several) with respect to task precedence relation. Moreover, it appends the additional row, where all tasks that belong to the critical path are duplicated. The example of the visualization with that feature enabled is shown in Fig. 6. One can notice that the critical path is saturated on the given image, which is indicated by the lack of gaps in the last row of visualization. From this fact we can infer that the solution shown in Fig. 6 is optimal, since the

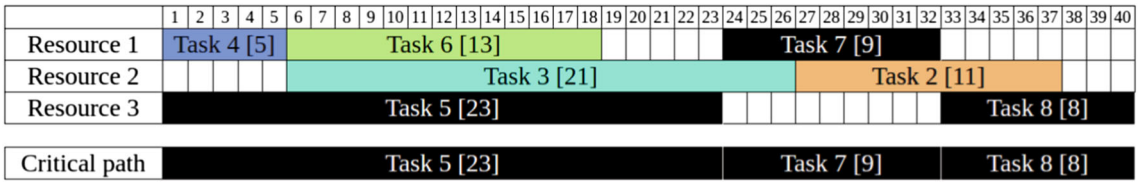


Fig. 6 Sample schedule visualization with critical path feature

Table 4 Greedy optimization of cost and duration

w_τ	w_τ=0.0				w_τ=1.0			
Instances	Best		Average		Best		Average	
	Cost	Duration	Cost	Duration	Cost	Duration	Cost	Duration
10_3_5_3	10845.3	149	10845.3	149	12992.7	108	11987.3	143
10_5_8_5	9013.6	154	9013.6	154	10736.4	100	10979.8	129
10_7_10_7	10215.2	149	10215.2	149	14153.0	104	15429.5	128
15_3_5_3	6289.5	239	6289.5	239	8297.0	230	9346.6	259
15_6_10_6	6946.2	233	6946.2	233	15040.7	131	14640	158
15_9_12_9	9841.5	155	9841.5	155	18049.6	90	17478.6	129

project cannot be executed in time shorter than the duration of its critical path.

5 Case studies

This section presents some applications of iMOPSE library: in didactic (see Sect. 5.1) to show efficiency of Greedy algorithm. It also describes the application in research—Greedy guided by Genetic Algorithm (see Sect. 5.2) and experimental application of Genetic Algorithm which uses weighted fitness function (see Sect. 5.3) in sampling two-dimensional solution space.

5.1 Didactic small instances: results of Greedy

Greedy schedule builder can be used to optimize either cost or duration. In case of cost, it is sufficient to choose the cheapest resource for each task, while disregarding duration. This approach always results in optimal solution cost-wise. Optimizing duration is not so trivial. First, random assignments are created and then start times of each tasks are set in Greedy manner. Consequence of this approach is the fact that the result is non-deterministic and depends on the initial task-resource assignment. Experiment has been performed by running both, cost and duration greedy optimization on a set of small, relatively simple data instances.

Table 4 shows a summary of a Greedy Algorithm (presented in Sect. 1) results using different values of w_τ. The value $w_\tau = 0.0$ means cost optimization and $w_\tau = 1.0$ means duration optimization. As expected, for each instance a better optimization value has been achieved for its respective optimization criteria. Interestingly, for instance 15_3_5_3 cost optimization has achieved the best (shortest one) duration very close to duration optimization and lower average duration. It is an anomaly caused by very specific set of resources in the dataset.

Instances from *EDU* dataset are "small" (consist of 10 or 15 tasks) and can be visualized and analyzed easily by a human. They can be solved by several methods. We used such instances (and *d36* dataset) in Wrocław University of Science and Technology in course entitled "Metaheuristics Solving Problems" to examine efficiency of heuristics and metaheuristics. Instances and provided *validator* allow us to organize the competitions for the best solution method presented by students. Since 2014 in each course edition such competition gives students extra motivation to learn metaheuristics.

We also used instances on "Artificial Intelligence and Expert Systems" course as a Constraint Satisfaction Problem solved by more computational demanding method like branch and bound.

5.2 Schedule optimization (cost vs duration) by Greedy guided by Genetic Algorithm

The purpose of this example is to show how approach deals with the problem. One of the advantages of a Genetic Algorithm in comparison with greedy approach is that intermediate weights can be used. In the experiment three weight configurations have been used: $w_\tau = 0.0$ for cost optimization; $w_\tau = 1.0$ for duration optimization and $w_\tau = 0.5$ for equal priority of each criteria.

Table 5 shows results achieved by GA for different evaluation weights. Since cost optimization is trivial, results are the same as the ones achieved by a greedy approach. Interestingly, durations achieved during pure cost optimization have

Table 5 Greedy guided by Genetic Algorithm optimization results according to various values of weight w_τ

w_τ	w_τ=0.0				w_τ=0.5				w_τ=1.0			
Instance	Best		Average		Best		Average		Best		Average	
	Cost	Duration	Cost	Duration	Cost	Duration	Cost	Duration	Cost	Duration	Cost	Duration
10_3_5_3	10845.3	149	10845.3	149	12622.2	93	12622.2	93	12622.2	93	12622.2	93
10_5_8_5	9013.6	154	9013.6	154	10802.2	100	10802.2	100	11148.5	100	11774.6	100
10_7_10_7	10215.2	131	10215.2	131	10731.2	104	10731.2	104	13423.1	104	14279.4	104
15_3_5_3	6289.5	263	6289.5	263	8297.0	230	8297.0	230	8297.0	230	9281.6	230
15_6_10_6	6946.2	216	6946.2	216	11109.7	102	11109.7	102	11989.5	102	14525.2	102
15_9_12_9	9841.5	155	9841.5	155	12292.2	90	12739.8	90	16375.7	90	18170.3	90

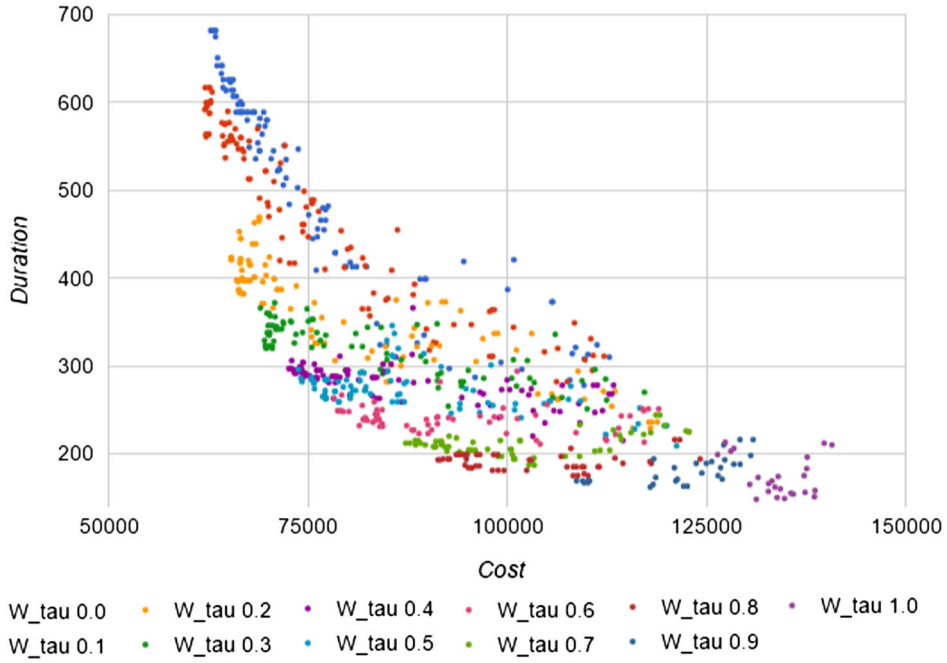


Fig. 7 Solution landscape (used 100_20_47_9 instance) estimated by DEGR (Myszkowski et al. 2018) using various values of w_τ

decreased in 2 cases and increased in 1 case. GA managed to improve results in 4 out of 6 cases for duration optimization. Balanced optimization has achieved durations equal to the ones achieved for duration optimization, while improving costs in 3 and matching in 2 out of 6 cases. Presumably, it has been caused by the simplicity of the dataset instances tested.

5.3 Weighted fitness function in MS-RCPSP

Provided iMOPSE library allows optimize both, cost and duration of the schedule. For that purpose, a weighted evaluator is used, which assigns weights to both criteria. Adjusting w_τ parameter lets user decide whether cost or duration is more important in the investigated approach. As example Hybrid Differential Evolution and Greedy Algorithm (DEGR) (Myszkowski et al. 2018) is used to examine how solution space is explored. To do it, DEGR has been executed separately using different values of w_τ where $w_\tau \in \{0.0, 0.1, \ldots, 1.0\}$. The gained solutions are collected and present in Fig. 7.

Weighted fitness function is quite simple as it projects two criteria into one value but can be used to estimate a shape of potential Pareto Front. Figure 7 depicts individuals that have been created during DEGR method with different value weights w_τ. Choosing only dominating individuals from this set creates a potential Pareto Front. Performing a similar process for every dataset instance is a first step to multi-criteria optimization. Fronts created in this manner can be used as a baseline, or to measure quality of the results from true multi-criteria approaches. DEGR approach is successfully examined using iMOPSE library (Myszkowski et al. 2017b) as quite effective method to get estimated Pareto Front.

6 Summary

In this paper we presented iMOPSE library that gives several useful tools connected to MS-RCPSP: not only instances *Generator*, *Validator* and *Visualizer* for MS-RCPSP instances but also two predefined datasets, for educational usage and experimental design. Also iMOPSE library consists of some extra Java code that implements Greedy Algorithm and Greedy Algorithm guided by Genetic Algorithm. It can be used as a didactic example or introduction for other researchers to use iMOPSE dataset.

The MS-RCPSP, in this work, is considered as bi-criteria optimization solved as simple weighted fitness function. We showed that it can be useful solution but we recommend MS-RCPSP solving as a multi-criteria problem and methods that uses dominance relation and Pareto Front [e.g., NSGA–II (Myszkowski et al. 2017b)]. The first results (see Myszkowski et al. 2017b; Wang and Zheng 2018) show that iMOPSE effectively supports bi-objective optimization and that is the main research direction of our future work.

The iMOPSE library has been launched in 2013 and is constantly used by our working group (see Skowroński et al. 2013b; Skowroński and Myszkowski 2013; Skowroński et al. 2013a; Myszkowski et al. 2015b, a; Myszkowski and Siemieński 2016; Myszkowski et al. 2018, 2017a, b). Each new work causes new tools to be developed, tested and published on iMOPSE homepage.

Acknowledgements The authors would like to thank the editor and the reviewers for their contribution in enhancing the technical quality of this paper.

Compliance with ethical standards

Conflict of interest Authors declare that he/she has no conflict of interest.

Ethical approval This article does not contain any studies with human participants or animals performed by any of the authors.

References

Błazewicz J, Lenstra JK, Rinnooy Kan AHG (1983) Scheduling subject to resource constraints: classification and complexity. Discrete Appl Math 5:11–24

Das PP, Acharyya S (2011) Simulated annealing variants for solving resource constrained project scheduling problem: a comparative study. In: Proceedings of 14th international conference on computer and information technology, pp 469–474

Hartmann S, Briskorn D (2010) A survey of variants and extensions of the resource-constrained project scheduling problem. Eur J Oper Res 207:1–14

Hartmann S, Kolisch R (2006) Experimental investigation of heuristics for resource-constrained project scheduling: an update. Eur J Oper Res 174:23–37

Kolisch R, Sprecher A (1996) PSPLIB—a project scheduling problem library. Eur J Oper Res 96:205–216

Luna F, Gonzalez-Alvarez DL, Chicano F, Vega-Rodriguez MA (2013) The software project scheduling problem: a scalability analysis of multi objective metaheuristics. Appl Soft Comput 15:136148

Myszkowski PB, Siemieński JJ (2016) GRASP applied to Multi-Skill Resource-Constrained Project Scheduling Problem. In: Computational collective intelligence, volume 9875 of the series lecture notes in computer science, pp 402–411

Myszkowski PB, Skowroński ME, Olech Ł, Oślizło K (2015a) Hybrid ant colony optimization in solving multi-skill resource-constrained project scheduling problem. Soft Comput J 19(12):3599–3619. https://doi.org/10.1007/s00500-014-1455-x

Myszkowski PB, Skowroński ME, Sikora K (2015b) A new benchmark dataset for Multi-Skill Resource-Constrained Project Scheduling Problem. In: Ganzha M, Maciaszek L, Paprzycki M (eds) Proceedings of the 2015 federated conference on computer science and information systems, ACSIS, vol 5, pp 129–138

Myszkowski PB, Laszczyk M, Kalinowski D (2017a) Co-evolutionary algorithm solving multi-skill resource-constrained project scheduling problem. In: Ganzha M, Maciaszek L, Paprzycki M (eds) Proceedings of the 2017 federated conference on computer science and information systems, ACSIS, vol 11, pp 75–82

Myszkowski PB, Laszczyk M, Lichodij J (2017b) Efficient selection operators in NSGA-II for solving bi-objective multi-skill resource-constrained project scheduling problem. In: Ganzha M, Maciaszek L, Paprzycki M (eds) Proceedings of the 2017 federated conference on computer science and information systems, ACSIS, vol 11, pp 83–86

Myszkowski PB, Olech LP, Laszczyk M, Skowroński ME (2018) Hybrid differential evolution and Greedy (DEGR) for solving multi-skill resource-constrained project scheduling problem. Appl Soft Comput 62:1–14

RESCON—homepage of project. http://feb.kuleuven.be/rescon/. Accessed 23 March 2017

Skowroński ME, Myszkowski PB (2013) Specialized genetic operators for multi-skill resource constrained project scheduling problem. In: 19th international conference on soft computing MENDEL, pp 57–62

Skowroński ME, Myszkowski PB, Kwiatek P, Adamski M (2013a) Tabu Search approach for Multi-Skill Resource-Constrained Project Scheduling Problem. In: Annals of computer science and information systems, vol 1, Proceedings of the 2013 federated conference on computer science and information systems, pp 153–158

Skowroński ME, Myszkowski PB, Podlodowski Ł (2013b) Novel heuristic solutions for Multi-Skill Resource-Constrained Project Scheduling Problem, Annals of computer science and information systems Volume 1, Proceedings of the 2013 federated conference on computer science and information systems, pp. 159–166,

Wang L, Zheng X-L (2018) A knowledge-guided multi-objective fruit fly optimization algorithm for the multi-skill resource constrained project scheduling problem. Swarm Evol Comput 38:54–63

Yannibelli V, Amandi A (2013) Hybridizing a multi-objective simulated annealing algorithm with a multi-objective evolutionary algorithm to solve a multi-objective project scheduling problem. Expert Syst Appl 40:2421–2434

Zhang H, Li H, Tam C (2006) Particle swarm optimization for resource-constrained project scheduling. Int J Project Manag 24:83–92

Zheng H-Y, Wang L, Zheng X-L (2017) Teachinglearning-based optimization algorithm for Multi-Skill Resource Constrained Project Scheduling Problem. Soft Comput 21(6):1537–1548

Ziarati K, Akbari R, Zeighami V (2011) On the performance of bee algorithms for resource-constrained project scheduling problem. Appl Soft Comput 11:3720–3733