Chapter 10
Proposed Metaheuristics for Solving Problem Θ_Z (DCSPwCRD)

10.1 IBEA—Island-Based Evolutionary Algorithm

An island-based evolution algorithm (IBEA) belongs to the class of distributed algorithms. To improve efficiency of genetic algorithms (GA) several distributed GA's were proposed in [1–3]. The proposed algorithms used an island-based approach where a set of independent populations of individuals evolves on "islands" cooperating with each other. The island-based approach brings two benefits: a model that maps easily onto the parallel hardware and extended search area (due to multiplicity of islands) preventing from sticking in local optima. Promising results of the island-based approach achieved in [4, 5] motivated the author to design the IBEA for discrete-continuous scheduling.

An island-based evolutionary algorithm (IBEA), proposed originally in [6], operates on two levels: on the island level and population level. To evolve individuals of the population level a population-based evolutionary algorithm (PBEA) is proposed. On the island level the following assumptions are made:

- all islands are located on a directed ring,
- an island is represented by a population of individuals,
- the populations of individuals evolve on each island independently,
- each island I_k regularly sends its best solution to the successor $I_{(k \bmod K)+1}$ in the ring, where $k = 1, 2, \ldots, K$, and K is the number of islands,

On the population level the following assumptions are made:

- an individual (a solution) is represented by an n-element vector $S = [c_i | 1 \leq i \leq n]$,

E. Ratajczak-Ropel and A. Skakovski, *Population-Based Approaches to the Resource-Constrained and Discrete-Continuous Scheduling*,
Studies in Systems, Decision and Control 108, DOI 10.1007/978-3-319-62893-6_10

- all processing modes of all tasks are numbered consecutively. Thus, processing mode l_b of task J_b has the number $c_b = \sum_{i=1}^{b-1} W_i + l_b$,
- all S representing feasible solutions are potential individuals,
- an initial population P_0 is composed from the potential individuals for whom task modes and tasks order on the list are random,
- each individual can be transformed into a schedule by applying LSG, which is a specially designed list-scheduling algorithm for discrete-continuous scheduling,
- each schedule produced by the LSG can be directly evaluated in terms of its fitness,
- new population is formed by applying several evolution operators: selection and transfer of some more "fit" individuals, random generation of individuals, crossover, and mutation,
- the algorithm stops when an optimality criterion is satisfied or the preset number of generations on each island have been generated,
- when IBEA stops, the best overall solution is the final one.

The following pseudo-code shows main stages of the IBEA algorithm:

```
Procedure IBEA
Begin
  Set number of islands K, number of generations PN and
  size of the population PS for each island.
  For each island I_k, generate an initial population P_0.
  While no stopping criteria is met do
    For each island I_k do
      Evolve PN generations on island I_k using PBEA.
      Send the best solution to I_(k mod K) + 1.
      Incorporate the best solution from I_((K+k-2) mod K)+1 in-
      stead of the best one.
  EndWhile.
  Find the best solution S_best across all islands and
  save it as the final one.
EndProc_IBEA.
```

The PBEA algorithm is shown in the following pseudo-code:

```
Procedure PBEA
Begin
 Set ic:= 0; (ic - iteration counter);
 While no stopping criteria is met do
  Set ic:= ic + 1
  Calculate fitness factor for each individual in population P_ic-1 using LSG;
  Form new population P_ic:
  Select randomly a quarter of PS of individuals from P_ic-1 (probability of selection depends on fitness of an individual);
  Create a quarter of PS of individuals by applying crossover operator to previously selected individuals from P_ic-1;
  Create a quarter of PS of individuals by applying mutation operators to previously selected individuals from P_ic-1;
  Generate a quarter of PS of individuals from the set of potential individuals (random task processing mode, and task order);
 EndWhile.
EndProc_PBEA.
```

The LSG algorithm used to transform S into a schedule is carried out as follows:

```
Procedure LSG
Begin
 Construct a list of tasks from the code representing an individual. Set loop over tasks on the list.
 Within the loop, allocate current task to a processor considering the amount of a continuous resource allocated to the task, and minimizing the beginning time of its processing. Continue with tasks until all have been allocated.
 Determine the fitness of individual S as Q_s = max{C_i}, i = 1, ... , n.
EndProc_LSG.
```

Table 10.1 The comparison of the results obtained by the IBEA and G_{dskr} for problem Θ_Z

Problem size n × m × W	Relative error (RE)	Discretisation level					
		$W = 10$		$W = 20$		$W = 50$	
		IBEA (%)	Gdskr (%)	IBEA (%)	G_{dskr} (%)	IBEA (%)	Gdskr (%)
10 × 2 × W	RE_{avg}	−3,09	7,59	−2,77	9,18	−2,86	10,41
	RE_{max}	50,02	15,34	49,09	15,84	52,22	17,54
10 × 3 × W	RE_{avg}	−0,45	14,58	−0,98	15,79	−0,14	17,11
	RE_{max}	73,33	25,18	69,65	23,15	71,91	27,52
20 × 2 × W	RE_{avg}	4,08	13,25	4,74	15,42	4,91	17,65
	RE_{max}	41,09	19,40	43,44	24,02	43,28	26,87

10.1.1 Computational Experiment

The proposed island-based evolution algorithm for solving discrete-continuous scheduling problems with continuous resource discretisation was implemented and tested. Results were compared to the best known obtained by a genetic GAVR-dyskr, tabu search, and simulated annealing algorithms [7] (GAVRdyskr, denoted as G_{dskr}, was used for results comparison as the one of the same nature).

Three combinations of $n \times m$ were considered (n—the number of tasks and m—the number of machines): 10 × 2, 10 × 3, and 20 × 2. For each $n \times m$ combination three discretisation levels were considered: 10, 20, and 50. For each discretisation level 100 instances of a problem Θ_Z were generated, which makes 900 instances of the problem. Each instance was tested 20 times. Relative error (RE) of the solutions found by the IBEA compared to best-known solutions was used to evaluate the quality of IBEA. RE calculated as $RE = (Q_{IBEA} - Q_{best\text{-}known})/Q_{best\text{-}known}$ for each instance was used to find average (RE_{avg}) and maximum (RE_{max}) relative errors. RE_{avg} and RE_{max} of the solutions found by the IBEA and G_{dskr} are presented in Table 10.1.

As it can be seen in Table 10.1 the quality of the solutions found by the IBEA is, on average, competitive with the quality of the solutions found by G_{dskr}. For example, for case 10 × 2, $W = 10$ $RE_{avg} = -3{,}09\%$, which means that the schedule length of all schedules yielded by IBEA was 3,09% shorter on average than the best-known. For the same case, $RE_{max} = 50{,}02\%$ means that the longest schedule among all schedules yielded by IBEA was 50,02% longer than the best-known. Such large RE_{max} points to the necessity of better tunning of the IBEA. As it can be also seen from Table 10.1, REs of the solutions do not always decrease as continuous resource discretisation level increases. Thus the level of continuous resource discretisation for which REs of the solutions are smallest should be determined empirically.

Mean time required by the IBEA to find a solution for 10×2 on CPU with Pentium III 733 MHz was 60 s. Mean time required by G_{dskr} to find a solution for 10×2 on supercomputer Silicon Graphics Power Challenge XL with twelve RISC MIPS R8000 processors was 33 s.

Such results make IBEA quite effective algorithm for solving discrete-continuous scheduling problems with continuous resource discretisation.

10.2 PLA—Population Learning Algorithm

Population learning algorithm (PLA) first introduced in [8] takes advantage of basic ideas, principals and assumptions introduced in a Social Learning Algorithm (SLA) originally proposed in [9]. Both SLA and PLA are based on an analogy to a social phenomenon rather than to evolutionary processes. Whereas evolutionary algorithms emulate basic features of natural evolution including natural selection, hereditary variations, the survival of the fittest and production of far more offspring than are necessary to replace current generation, the population learning algorithms take advantage of features that are common to social education systems:

- a generation of individuals enters the system,
- individuals learn through organized tuition, interaction, self-study and self-improvement,
- learning process is inherently parallel with different schools, curricula, teachers, etc.,
- learning process is divided into stages,
- more advanced and more demanding stages are entered by a diminishing number of individuals from the initial population (generation),
- at higher stages more advanced education techniques are used,
- the final stage can be reached by only a fraction of the initial population.

In the PLA, the assumptions made for the individuals are the same as the assumptions made for the individuals of the IBEA on the population level, see Sect. 10.1.

Initially, a number of individuals, known as the initial population, is randomly generated. Once the initial population has been generated, individuals enter the first learning stage. It involves applying some, possibly basic and elementary, improvement schemes. These can be based, for example, on some simple local search procedures. The improved individuals are then evaluated and better ones pass to the subsequent stage. A strategy of selecting better or more promising individuals must be defined and duly applied. In the following stages the whole

cycle is repeated. Individuals are subject to improvement and learning, either individually or through information exchange, and the selected ones are again promoted to the higher stage with the remaining ones dropped-out from the process. At the final stage the remaining individuals are reviewed and the best represents a solution to the problem at hand.

The PLA is seen here as a general framework for constructing hybrid solutions to difficult computational problems. Strength of the PLA stems from integrating in an "intelligent" manner the power of population-based algorithms using some random mechanism for diversity assurance, with efficiency of various local search algorithms. The later may include, for example, reactive search, tabu search, simulated annealing as well as the described earlier population based approaches.

General idea of the present implementation of the PLA proposed in [8] is shown in the following pseudo code:

```
Procedure PLA
Begin
  Generate an initial population P0 of size x0, where
  task order and modes in each individual are random.
  Select x1 most promising individuals from P0 to P1,
  where x1 << x0.
  Learn all individuals in P1 with aid of procedure
  IBEA.
  Select the best individual (BI) from P1 to the last
  learning stage.
  Learn BI with the aid of procedure TS.
  Output the best solution to the problem.
EndProc_PLA.
```

In the presented pseudo code, procedure IBEA stands for the Island-Based Evolution Algorithm described in Sect. 10.1, and TS—for Tabu Search, which is to be presented in the subsequent section.

10.2.1 Tabu Search

Tabu search is another metaheuristic used in the considered PLA (see [10]). In order to present the general idea of present implementation of the tabu search procedure, we introduce the neighborhoods N_t and N_{md} of a solution S. N_t is a set of solutions generated from S by moving task $J_i \in S$ from place i to the rest $n-1$ places. Thus, we yield $|N_t| = n \cdot (n-1)$ neighbors. N_{md} is a set of solutions generated from S by assigning to task $J_i \in S$ one by one in a row all of its W modes, assuming that all tasks can be executed in W modes. Thus we yield another $|N_{md}| = n \cdot (W-1)$ neighbors. The considered Tabu Search procedure is shown in the following pseudo code:

```
Procedure TS
Begin
  Set S_0 = S_I, where S_I = BI from P_1.
  Set the best solution S_best = S_0.
  Set Tabu List TL = {∅}.
  Set N_t = {S_I} and N_md = {∅}.
  Repeat the following max_number_of_iterations times:
    Find the best legal neighbour S_bln of S_0, i.e. the
    best across N_t and N_md neighbour which is not on TL.
    Set S_0 = S_bln.
    If S_bln is more fit than S_best then S_best = S_bln.
    Put S_bln on the Tabu list.
    If the fitness of S_0 has not improved after n_it num-
    ber of iterations construct a new solution by moving
    task J_i in S_0 to one of the chosen randomly less fre-
    quently visited places on the task list and assign-
    ing to it one of the chosen randomly less frequently
    assigned execution modes.
  EndRepeat.
EndProc_TS.
```

The size of the Tabu List (TL) was determined empirically and set to 500 solutions.

10.2.2 Computational Experiment

The proposed population learning algorithm for solving discrete-continuous scheduling problems with continuous resource discretisation was implemented and tested. Results were compared to the best known obtained by a genetic GAVRdyskr, tabu search, simulated annealing algorithms [7], and the IBEA described in [4] (GAVRdyskr, denoted in the rest of the text as G_{dskr}, and the IBEA were used for results comparison in as the ones of the same nature). For testing purposes three combinations of $n \times m$ were considered (n—the number of tasks and m—the number of machines): 10×2, 10×3, and 20×2. For each $n \times m$ combination three discretisation levels were considered: 10, 20, and 50. For each discretisation level 100 instances of a problem Θ_Z were generated, which makes 900 instances of the problem. The instances of the problem were generated with aid of a procedure received from the author of [7]. Each instance was tested 26 times. Relative error (RE) of the solutions found by the PLA compared to best-known solutions was used to evaluate the quality of the PLA.

The value of the RE calculated for each instance according to the formulae $RE = (Q_{PLA} - Q_{best\text{-}known})/Q_{best\text{-}known}$ was used to find average (RE_{avg}) and maximum (RE_{max}) relative errors. RE_{avg} and RE_{max} of the solutions found by the PLA, IBEA, and G_{dskr} are presented in Table 10.2. As it can be seen in Table 10.2, the quality of the solutions found by the PLA has been, on average, better than the quality of the solutions found by the IBEA and G_{dskr}. For example, for case 10×2, $W = 10$, $RE_{avg} = -2{,}12\%$, which means that the schedule length of all schedules yielded by the PLA was 2,12% on average shorter than the best-known. For the same case, $RE_{max} = 56{,}34\%$ means that the longest schedule among all schedules yielded by PLA was 56,34% longer than the best-known. Such large RE_{max} points to the necessity of better tuning of the PLA. Despite large RE_{max}, the PLA improved 55% of 300 the best known solutions for sizes 10×2, 10×3 and 20×2, and reduced average values of RE_{avg} compared to the IBEA and G_{dskr}. Table 10.3 shows the average of the PLA's RE_{avg} reduction percentage in comparison to average RE_{avg} of the IBEA and G_{dskr}. In other words, Table 10.3 shows how many percent on average the solutions found by the PLA were better than the solutions found by the IBEA and G_{dskr}. As it can be also seen from Table 10.2, REs of the solutions do not always decrease as continuous resource discretisation level increases. Thus, the level of continuous resource discretisation for which REs of the solutions are smallest should be determined empirically.

Table 10.2 The comparison of REs obtained by the PLA, IBEA and G_{dskr} for problem Θ_Z

Problem size $n \times m \times W$	Relative error (RE)	Discretisation level								
		$W = 10$			$W = 20$			$W = 50$		
		PLA (%)	IBEA (%)	G_{dskr} (%)	PLA (%)	IBEA (%)	G_{dskr} (%)	PLA (%)	IBEA (%)	G_{dskr} (%)
$10 \times 2 \times W$	RE_{avg}	−2,12	−0,61	7,59	−1,44	0,19	9,18	−1,07	0,36	10,41
	RE_{max}	56,34	62,35	15,34	56,69	62,42	15,84	61,55	62,88	17,54
$10 \times 3 \times W$	RE_{avg}	1,46	2,89	14,58	1,03	3,49	15,79	1,66	5,22	17,11
	RE_{max}	77,51	83,11	25,18	76,17	84,91	23,15	78,80	82,40	27,52
$20 \times 2 \times W$	RE_{avg}	5,62	6,62	13,25	6,27	6,96	15,42	6,59	7,13	17,65
	RE_{max}	46,50	47,84	19,40	47,60	47,8	24,02	47,83	47,86	26,87

Table 10.3 The average of the PLA's RE_{avg} reduction percentage in comparison to average RE_{avg} of the IBEA and G_{dskr}

Problem size $n \times m \times W$	IBEA (%)	G_{dskr} (%)
$10 \times 2 \times W$	1,52	10,60
$10 \times 3 \times W$	2,48	14,44
$20 \times 2 \times W$	0,74	9,28

Mean time required by the PLA to find a solution for 10×2 on Pentium (R) 4 CPU 3.00 GHz compiled with aid of Borland Delphi Personal v.7.0 was 5 s, by IBEA compiled in Borland Pascal v.7.0—48 s. Mean time required by G_{dskr} to find a solution for 10×2 on supercomputer Silicon Graphics Power Challenge XL with twelve RISC MIPS R8000 processors was 33 s. Such results make PLA quite effective algorithm for solving discrete-continuous scheduling problems with continuous resource discretisation.

10.3 PLA2—Cross-Entropy-Based Population Learning Algorithm

A cross-entropy-based population learning algorithm (PLA2) proposed in [11] is another attempt to make use of the idea of social learning framework already presented in Sect. 10.2. Different to the PLA structure, more advanced procedure for the initial population creation and different setting for the TS procedure contributed to higher efficiency of the PLA2.

In the PLA2, the assumptions made for the individuals are the same as the assumptions made for the individuals of the IBEA on the population level, see Sect. 10.1. The general idea of the implementation of the PLA2 is shown in the following pseudo code:

```
Procedure PLA2
Begin
  Create an initial population P0 of the size x0 - 1 using procedure cross-entropy (CE).
  Create an individual TSI in which all tasks Ji are to be executed in mode li = 1 (a mode characterized by minimal quantity of additional resource u_i^1 and maximal task processing time τ_i^1, 1 ≤ i ≤ n).
  Improve individual TSI with Tabu Search (TS) procedure.
  Create population P1 = P0 + TSI.
  Improve all individuals in P1 with IBEA.
  Output the best solution to the problem.
EndProc_PLA2.
```

In the description of the procedure PLA2 above, $x_0 = K \cdot PS$, where K—the number of islands and PS—the population size on an island defined in procedure IBEA. As it follows from the description of the PLA2, population P_1 comprises the initial population for the IBEA, therefore the step for generating the initial population for the IBEA, as it is given in the description of the IBEA in Sect. 10.1, should be omitted. The description of TS procedure can be found in Sect. 10.2.1.

10.3.1 Cross-Entropy Algorithm

A cross-entropy (CE) procedure, proposed in the PLA2, is perceived as the procedure for preparing some solution basis for further improvement by procedure IBEA. In CE procedure a cross-entropy method first proposed in [12] is used since it was effective in solving various difficult combinatorial optimization problems [13]. Because in CE procedure a solution is viewed as a vector of n tasks, we would like to know the probability of locating task J_i on a particular place j in the vector. For this reason we introduce two success probability vectors $\hat{p}_j$ and $\hat{p}'_{ji}$ related to each task J_i and its place j in solution S. Vector $\hat{p}_j = \{p_{ji} \mid 1 \le i \le n\}$, $1 \le j \le n$ contains p_{ji} values, which is the probability that on place j there will be located task i. Vector $\hat{p}'_{ji} = \{p_{jil} \mid 1 \le l \le W_i\}$, $1 \le j \le n$, $1 \le i \le n$ contains p_{jil} values, which is the probability that on place j task i will be executed in mode l. A procedure CE using cross-entropy method for combinatorial optimization described in [13] and modified for solving problem Θ_Z is shown in the following pseudo code:

Procedure CE

Begin

Set ic:= 1 (ic – iteration counter), ic^{stop} – maximal number of iterations, a:= 1.

Set $$\hat{p}_j = \{p_{ji} = 1/n \mid 1 \le i \le n\}, 1 \le j \le n. \tag{10.1}$$

Set $$\hat{p}'_{ji} = \{p_{jil} = 1/W_i \mid 1 \le l \le W_i\}, 1 \le j \le n, 1 \le i \le n. \tag{10.2}$$

While $ic \le ic^{stop}$ do

Generate a sample S_1, S_2, … , S_s, … , S_N of solutions with success probability vectors $\hat{p}_j$ and $\hat{p}'_{ji}$.

Order S_1, S_2, … , S_s, … , S_N by nondecreasing values of their fitness function.

Set $$\gamma = \lceil \rho \cdot N \rceil, \rho \in (0,1). \tag{10.3}$$

Set $$\hat{p}_j = \left\{ p_{ji} = \frac{\sum_{s=1}^{\gamma} I(S_s(j) = i)}{\gamma} \mid 1 \le i \le n \right\}, \tag{10.4}$$

$1 \le j \le n$, $I(S_s(j) = i) = 1$, $I(S_s(j) \ne i) = 0$, where $S_s(j)$ – the number of the task located on j-th place in s-th solution S.

Set $$\hat{p}'_{ji} = \left\{ p_{jil} = \frac{\sum_{s=1}^{\gamma} I(S_s(ji) = l)}{\gamma} \mid 1 \le l \le W_i \right\}, \tag{10.5}$$

$1 \le j \le n$, $1 \le i \le n$, $I(S_s(ji) = l) = 1$, $I(S_s(ji) \ne l) = 0$, where $S_s(ji)$ – an execution mode of task i located on j-th place in s-th solution S.

Save the first $h = \lceil K \cdot PS / ic^{stop} \rceil$ best solutions from the ordered sample into P_0 under address a. Set a:= a + h.

Set ic:= ic + 1.

EndWhile.

EndProc_CE.

In the presented pseudo code, a parameter N is the number of solutions in a sample generated in each iteration. A parameter ρ determines the percentage of the best solutions in the current sample that are used to calculate new values for vectors $\hat{p}_j$ and $\hat{p}'_{ji}$. Both parameters were determined empirically and set $N = 1000$ and $\rho = 0{,}2$. Parameters K—the number of islands and PS—the population size are defined in procedure IBEA and PBEA respectively.

10.3.2 Computational Experiment

The considered cross-entropy-based population learning algorithm for solving discrete-continuous scheduling problems with continuous resource discretisation (PLA2), was implemented and tested. Results were compared to the best known obtained by a genetic GAVRdyskr, tabu search, simulated annealing algorithms [7], IBEA [4], and PLA [8] (GAVRdyskr is denoted in the rest of the text as G_{dskr}).

10.3.2.1 Assumptions of the Experiment

For the testing purposes, the following set of assumptions has been formulated and implemented:

- all tasks J_i can be processed in W modes, i.e. $W_i = W$, $i = 1, 2, \ldots, n$, where $W \in \{10, 20, 50\}$.
- continuous resource is discretised uniformly and the amount of the continuous resource assigned to task J_i in mode l_i can be calculated as:

$$u_i^{l_i} = \frac{l_i}{W_i}, \quad l_i = 1, 2, \ldots, W_i, \quad i = 1, 2, \ldots, n \tag{10.6}$$

- task processing rate function f_i is concave and its value can be calculated as:

$$f_i = u_i^{l_i 1/\alpha_i}, \quad \alpha_i \in \{1, 2\}, \quad i = 1, 2, \ldots, n \tag{10.7}$$

- the processing time of task J_i in mode $l_i = 1, 2, \ldots, W_i$ can be calculated as:

$$\tau_i^{l_i} = \frac{\tilde{x}_i}{f_i}, \; i = 1, 2, \ldots, n \tag{10.8}$$

- task sizes $\tilde{x}_i$, $i = 1, 2, \ldots, n$ were generated from interval [1, 1000] with uniform probability distribution,
- the number of tasks $n \in \{10, 20\}$ and the number of machines $m \in \{2, 3\}$.

For the testing purposes three combinations of $n \times m$ were considered—10×2, 10×3, and 20×2. For each considered combination of $n \times m$, 100 instances of problem Θ_Z, have been generated, which together with three discretisation levels (10, 20, 50) makes available 900 instances of the problem. Each instance was tested 24 times. A relative error (RE) of the solutions found by PLA2 with respect to best-known solutions was used to evaluate the quality of PLA2. The value of RE calculated using equation $RE = (Q_{PLA2} - Q_{best\text{-}known})/Q_{best\text{-}known}$ for each instance was used to find average and maximum relative errors.

10.3.2.2 Fine Tuning of PLA2

All three procedures used in the PLA2 have a stochastic nature. The lack of mathematically proved rules makes it difficult to deduce values of the parameters used by the respective procedures. Instead, these values have to be set experimentally. An extensive computational experiment covered all three procedures. In each case we have been looking for the most efficient settings, that is such settings that have been yielding highest quality solutions within some pre-set number of the fitness function evaluations. We tried not only to choose the most proper parameters of the algorithm, but also to find the sequence of the learning stages such that quality of the found solutions was highest possible. Because PLA2 is one more attempt to use an evolutionary approach for coping with the discrete-continuous scheduling problem our goal was to achieve better results for the same number of the fitness function evaluations equal 720000 which was used in the previous trials. While determining the most proper sequence of the learning stages we considered two test versions of PLA2. In these two test versions, the primary solutions were generated according to the rule: random task order in a solution and random task processing mode. In the first test version, primary solutions were improved on the second stage by the IBEA, then on the third stage the best found by the IBEA solution was improved by procedure TS. However, we implemented in PLA2 more efficient second version, in which on the second stage, there was generated only one solution by procedure TS and next added to the set of the primary solutions found on the first stage. This way obtained set of solutions was improved on the third stage by the IBEA. The second version improved 64,67% of 300 best known solutions, while the first—57%, and the average of RE_{avg} was respectively 2,85% and 3,04%. In the final version of the PLA2 we replaced simple first stage procedure, which was used in the PLA for generating the set of the primary solutions, by procedure CE. The percent distribution of the whole number of fitness function evaluations among particular learning procedures is as follows: CE—7%, TS—30%, IBEA—63%. The further tuning concerned the particular procedures engaged in the PLA2.

Procedure CE used to yield the set of primary solutions instead of their random generating increased the percentage of the improved best known solutions from 64,67 to 80,33%, and decreased average of RE_{avg} from 2,85 to 2,09%. For test purposes we designed two versions of procedure CE—with accumulation of the values of success probability vectors $\hat{p}_j$ and $\hat{p}'_{ji}$ in each iteration, and without accumulation. In version with accumulation we calculated the values of vectors $\hat{p}_j$ and $\hat{p}'_{ji}$ in iteration $ic + 1$ using equation:

$$\hat{p}_j(ic+1) = \hat{p}_j(ic-1) + \hat{p}_j(ic) \tag{10.9}$$

$$\hat{p}'_{ji}(ic+1) = \hat{p}'_{ji}(ic-1) + \hat{p}'_{ji}(ic) \tag{10.10}$$

and in the version without accumulation as:

$$\hat{p}_j(ic+1) = \hat{p}_j(ic) \tag{10.11}$$

$$\hat{p}'_{ji}(ic+1) = \hat{p}'_{ji}(ic) \tag{10.12}$$

where values of $\hat{p}_j(ic)$ and $\hat{p}'_{ji}(ic)$ vectors were calculated on the basis of the best γ solutions generated in iteration ic. The main disadvantage of the version with accumulation was sticking in local optima, which explains including the version without accumulation into the final version of the PLA2. In the final implementation of CE procedure we set sample size $N = 1000$ and $\rho = 0{,}2$.

While tuning TS procedure we also considered two versions of it. In both versions of the procedure we generated "a task neighbourhood" set of solutions of a solution S by moving a task $J_i \in S$ from place i to the rest $n - 1$ places in S. In the first version of TS procedure we generated an additional "mode neighbourhood" set of S by assigning to each task $J_i \in S$ one by one in a row the rest all of its $W - 1$ modes, assuming that all tasks can be executed in W modes. The best solution of the iteration was determined as the best across both sets. In the second version of TS procedure, while generating "task neighbourhood" set after each move, we "tuned" the mode only of a single just moved task by assigning to it one by one in a row its remaining $W - 1$ modes. The best solution of the iteration was determined as the best among all solutions generated in such manner. In the final version of the PLA2 we implemented the first TS procedure version as more efficient.

Fine tuning of the IBEA procedure focused on finding the most effective combination of the parameter values. It is known from the literature, that parameters which have a direct impact on the efficiency of the island model are: the number of islands, the size of population on an island [14, 15], migration size [16, 17], migration interval [17, 18], migration policy [16], migration topology [19], and the heterogeneity of the island model [20]. We restricted ourselves to determining on the way of experiment the number of islands K, the best solutions frequency exchange between islands xfq, stop criterion ic^{stop}, and population size on an island PS. The considered parameters were set respectively: $K = 15$, $xfq = 3$, i.e. islands exchanged their best solutions after three populations of individuals were generated

on each island, $ic^{stop} = 2000$, i.e. IBEA stopped after 2000 populations had been generated on each island, $PS = 24$ individuals in a population. While tuning IBEA we also considered selection of individuals from the previous population and generating "wild" individuals to the next population. The probability of selection of individuals from previous to the next population depended on the value of the fitness function. In the present PLA2 implementation, tasks' places and tasks' processing modes in a solution vector were determined according to the uniform distribution. Crossover and mutation operators were applied to individuals selected from the previous population. Two individuals took part in each crossover with number of "genes" that were exchanged between parents chosen at random. Mutation of the selected individuals was performed in three ways with probability 0,25, 0,5 and 0,25 respectively. In the first type of mutation we changed at random task processing mode of a chosen at random task. In the second type—chosen at random task J_i was swapped with task J_{i+1}. In the third type of mutation two chosen at random tasks were swapped. All auxiliary random values used in the crossover and mutation operators were acquired according to the uniform probability distribution.

10.3.2.3 Results of the Experiment

RE_{avg} and RE_{max} of the solutions found by the PLA2, PLA, and G_{dskr} are presented in Table 10.4.

The quality of the solutions found by the PLA2 is, on average, better than the quality of the solutions found by the PLA and G_{dskr}. For example, for case 20×2, $W = 20$ $RE_{avg} = -0{,}23\%$, which means that the schedule length of all schedules yielded by the PLA2 was, on average, 0,23% shorter than the best-known. For the same case, $RE_{max} = 7{,}23\%$ means that the longest schedule among all schedules yielded by the PLA2 was 7,23% longer than the best-known. In our tests the PLA2 improved 80,33% of 300 the best known solutions for combinations 10×2, 10×3, and 20×2, and reduced average of RE_{avg} compared to the PLA and G_{dskr}. Table 10.5 shows how many percent on average the solutions found by the PLA2 were better than the solutions found by the PLA and G_{dskr}.

Mean time required by the PLA2 and the PLA to find a solution for 10×2 on Pentium (R) 4 CPU 3,00 GHz compiled with aid of Borland Delphi Personal v.7.0 was 5 s. The mean time required by G_{dskr}, which was implemented in C++, to find a solution for 10×2 on supercomputer Silicon Graphics Power Challenge XL designed in 64-bit SMP (Symmetrical Multi Processing) architecture on 12 RISC MIPS R8000 processors using 1 GB RAM and 20 GB disc memory was 33 s. Such results make the PLA2 quite effective algorithm for solving problem Θ_Z.

Table 10.4 Comparison of REs obtained by the PLA2, PLA and G_{dskr} for problem Θ_Z

Problem size $n \times m \times W$	Relative error (RE)	Discretisation level								
		$W = 10$			$W = 20$			$W = 50$		
		PLA2 (%)	PLA (%)	Gdskr (%)	PLA2 (%)	PLA (%)	Gdskr (%)	PLA2 (%)	PLA (%)	Gdskr (%)
$10 \times 2 \times W$	RE_{avg}	2,75	1,82	7,59	1,53	2,52	9,18	2,25	2,93	10,41
	RE_{max}	9,29	9,39	15,34	5,94	9,00	15,84	8,32	12,21	17,54
$10 \times 3 \times W$	RE_{avg}	2,99	3,76	14,58	2,20	3,32	15,79	2,19	3,96	17,11
	RE_{max}	10,66	11,58	25,18	8,70	12,40	23,15	33,54	14,31	27,52
$20 \times 2 \times W$	RE_{avg}	2,70	2,49	13,25	−0,23	3,11	15,42	2,44	3,42	17,65
	RE_{max}	9,21	9,48	19,40	7,23	9,65	24,02	9,03	9,65	26,87

Table 10.5 The average of the PLA2's RE_{avg} reduction compared to average RE_{avg} of the PLA and G_{dskr} given in percent

Problem size $n \times m \times W$	PLA (%)	G_{dskr} (%)
$10 \times 2 \times W$	0,25	6,89
$10 \times 3 \times W$	1,22	13,36
$20 \times 2 \times W$	1,37	13,80

10.4 PLA3—Population Learning with Differential Evolution Algorithm

The PLA model can be also viewed as an island model, were islands are connected to each other according to some topology and exchange individuals in order to collectively find best possible solution to the problem. We used four learning procedures to design the PLA3: Cross-Entropy (CE), Differential Evolution (DE), Tabu Search (TS), and a Population-Based Evolutionary Algorithm (PBEA) [21]. The PLA3 extends the earlier designed PLA2 [11] with help of the Differential Evolution (DE) method, first proposed in [22]. The PLA3 inherits from the most efficient version of PLA2 (denoted as AX-m in [23]) heterogeneity of the islands, on which diverse learning procedures are realized, random interconnection structure, and solution exchange adjusted to the specificness of the heterogeneous islands. We will use the terms *learning procedure* and an *island* interchangeably in the rest of the text.

We distinguish two categories of island groups—heterogeneous and homogeneous, dependently on the type of the learning procedures carried out on the islands. We refer to the group of islands as heterogeneous, if the learning procedure carried out on at least one island is different from the learning procedures carried out on the rest of the islands in the group. We refer to the group of islands as homogeneous, if the same learning procedure is carried out on each island in the group. In our work, we will refer to a particular island as heterogeneous (Ht), if TS or CE or DE procedure is carried out on it, and we will refer to an island as a homogeneous (Hm), if the PBEA is carried out on it. Because the PLA3 is the extension of its predecessor PLA2, it inherits among others the solution exchange policy. The solution exchange in the PLA3 is carried out among randomly chosen islands and is inherited from AX-m, which, according to [22], is the most efficient version of PLA2. In the PLA3, the assumptions made for the individuals are the same as the assumptions made for the individuals of the IBEA on the population level, see Sect. 10.1. The pseudo code of the proposed PLA3 is given below.

```
Procedure PLA3
Begin
  Set K ≥ 3 - the number of Hm-islands. Assign PBEA pro-
  cedure to all Hm-islands.
  Assign procedures to Ht-islands: TS procedure to Ht1,
  CE procedure to Ht2, DE procedure to Ht3.
  Set xHm - the total amount of solutions in the initial
  population PHm on all Hm-islands.
  Create an initial population PCE of size N ≥ xHm for
  Ht2 using Cross-Entropy procedure (CE).
  Send the best xHm solutions from PCE to PHm.
  Distribute equally individuals from PHm among all
  Hm-islands.
  Create an individual TSI on Ht1. In TSI all tasks Ji
  are to be executed in mode li = 1 (a mode character-
  ized by minimal quantity of additional resource u_i^1
  and maximal task processing time τ_i^1, 1 ≤ i ≤ n).
  Create an initial population PDE of size xDE using
  Cross-Entropy procedure (CE) on Ht3.
  Improve individuals on all islands using assigned to
  them procedures, cyclically exchanging best solutions
  among randomly chosen islands.
  Output the best solution to the problem.
EndProc_PLA3.
```

The solution exchange among the islands occurs after all islands have carried out the preset number of solution evaluations. Generally, the solution exchange is carried out between a pair of islands chosen at random from all available islands. However, there are some exceptions from that rule. In the PLA3 procedure, we distinguish several cases of the solution exchange which are described as follows.

In the case when the solution exchange is carried out between two randomly chosen homogeneous islands Hm_{r1} (PBEA procedure) and Hm_{r2} (PBEA procedure), each island in pair sends to the other one its best current solution.

When the exchange is carried out between Ht_1 island (TS procedure) and Hm_r island, as well as between Ht_1 island and Ht_3 island (DE procedure), each island in pair sends to the other one its best current solution. The exchange procedure between Ht_1 island and Ht_2 island (CE procedure) is described in the next paragraph.

When Ht_2 island (CE procedure) participates in the exchange, the transfer of the solutions is asymmetric. From all available islands Ht_2 receives $\gamma_{CE} = \rho_{CE} \cdot N$ solutions in total, where N is the population size on the Ht_2 island, and $\rho_{CE} = 0{,}2$. In this case of exchange, Ht_2 receives K the best current solutions from all Hm islands, ten copies of the best current solution from Ht_1 island, and $\min(\gamma_{CE} - K - 10, x_{\mathrm{DE}})$ best current solutions from Ht_3 island (DE procedure). If $x_{\mathrm{DE}} < \gamma_{CE} - K - 10$, then Ht_2 additionally receives $\gamma_{CE} - K - 10 - x_{\mathrm{DE}}$ solutions from a randomly chosen island Hm_r and next to it consecutive islands Hm_{r+1}, Hm_{r+2}, …, and Hm_{r+q} until the total number of the solutions received by Ht_2 is equal to γ_{CE}. When $r + q > K$, the numbering of the following consecutive Hm-islands starts with the island number 1. On the other hand, the transfer of the solutions from Ht_2 to the rest of the islands is carried out as follows. When the randomly chosen island in pair is Hm_r, Ht_2 sends its best PS current solutions to it, where PS is the population size on every Hm-island, defined in PBEA procedure. When the other island in pair is Ht_1, Ht_2 sends its best current solution to it. When the other island in pair is Ht_3, Ht_2 sends its best $\min(\gamma_{CE}, \gamma_{DE-CE})$ current solutions to Ht_3 island, where the value of $\gamma_{DE-CE} = \rho_{DE-CE} \cdot x_{\mathrm{DE}}$, and $\rho_{DE-CE} = 0{,}333$. These solutions substitute the worst solutions on Ht_3 island.

When the exchange is carried out between Ht_3 island (DE procedure) and Hm_r island, they exchange their $\min(PS, \gamma_{DE-Hm})$ best current solutions, where PS is the population size on every Hm-island and $\gamma_{DE-Hm} = \rho_{DE-Hm} \cdot x_{\mathrm{DE}}$, $\rho_{DE-Hm} = 0{,}333$. The solution exchange between Ht_3 and Ht_1, and Ht_3 and Ht_2 islands is described in two previous paragraphs.

10.4.1 Computational Experiment

The proposed population learning algorithm PLA3 for solving discrete-continuous scheduling problems with continuous resource discretisation was implemented and tested. There were 12 islands used in the PLA3 altogether, namely, the number of homogeneous islands was set to $K = 9$ with the PBEA procedure assigned to them and 3 heterogeneous islands. TS procedure was carried out on Ht_1, CE procedure on Ht_2, and DE procedure on Ht_3 island. The size of the population on every Hm-island was set to $PS = 24$. In the TS procedure, the size of the Tabu List (TL) was set to 500 solutions. In the CE procedure, parameters N and ρ_{CE} were set $N = 1000$ and $\rho = 0{,}2$ respectively. The size of the population on Ht_3 island was

set $x_{DE} = 2000$, which is different from the sizes considered in [24], where x_{DE} $\{20, 40, 80, 60, 100\}$. The rest of the parameters necessary to carry out the differential evolution algorithm were set to the same values as in [24], namely the scale factor A which controls the evolution rate of the population was set $A = 1,5$ and the values of the variable $rand \in [0, 1]$. The crossover constants Cr_p and Cr_m which control the probability that the trial individual will receive the actual individual's genes were set $Cr_p = 0{,}2$ and $Cr_m = 0{,}1$, where p and m in the notations Cr_p and Cr_m stand for tasks' positions and modes. For testing purposes three combinations of $n \times m$ were considered (n—the number of tasks and m—the number of machines): 10×2, 10×3, and 20×2. For each combination $n \times m$, 100 instances of a problem Θ_Z were generated and three discretisation levels W were considered: 10, 20, and 50. This way we considered nine sizes of the problem: $10 \times 2 \times 10$, $10 \times 2 \times 20$, $10 \times 2 \times 50$, $10 \times 3 \times 10$, …, $20 \times 2 \times 50$, which makes 900 instances of the problem in total. Each instance was tested 43 times. Mean time required by the PLA3 to find a solution for the problem sizes 10×2 and 10×3 for all discretisation levels on a PC under 64-bit operating system Windows 7 Enterprise with Intel(R) Core(TM) i5-2300 CPU @ 2,80 GHz 3,00 GHz, RAM 4 GB compiled with aid of Borland Turbo Delphi for Win32 was approximately 2–3 s, and for the problem size 20×2 for all discretisation levels approximately 4–6 s.

In order to evaluate the efficiency of the PLA3, we have used three types of relative errors: minimum, average, and maximum relative error of the solutions yielded by the algorithm. Relative errors (REs) of the solutions compared to the best-known solutions were calculated according to the formulae $RE = (Q_{algm} - Q_{best\text{-}known})/Q_{best\text{-}known}$, where Q_{algm}, $Q_{best\text{-}known}$—the schedule length of a solution found by the considered algorithm and the best-known solution respectively. The set of the best-known solutions was determined by the authors while using all designed by them algorithms and procedures for solving problem Θ_Z. We have determined RE_{max} for every size of the considered problem as a maximum RE across 4300 REs calculated while solving 100 instances, run 43 times each. We have also determined RE_{avg} as a mean value of 4300 REs obtained within 43 runs of 100 instances of the considered problem. We have compared the REs of the solutions found by the PLA3 to the REs of the solutions found by AX-m—the most efficient version of the PLA2 described in [23]. The values of RE_{avg} and RE_{max} for the PLA3 and AX-m (PLA2) for all problem sizes are presented in Table 10.6.

The values of REs in Table 10.6 show how much schedules yielded by the PLA3 were longer than the best known schedule for the same case. For example, for the case $10 \times 2 \times 10$ $RE_{avg} = 2{,}70\%$ means that the schedule length of all schedules yielded by the PLA3 was on average 2,70% longer than the best-known. For the same case, $RE_{max} = 10{,}14\%$ means that the longest schedule among all schedules yielded by the PLA3 was 10,14% longer than the best-known.

Table 10.6 The comparison of the relative errors of solutions found by the PLA3 and AX-m (the most efficient version of the PLA2) for the problem Θ_z

Problem size n × m × W	Relative error (RE)	Discretisation level					
		$W = 10$		$W = 20$		$W = 50$	
		PLA3 (%)	AX-m (%)	PLA3 (%)	AX-m (%)	PLA3 (%)	AX-m (%)
10 × 2 × W	RE_{min}	0,01	0,01	**−0,42**	0,00	**−0,49**	0,00
	RE_{avg}	**2,70**	3,54	**1,53**	2,27	**1,81**	2,47
	RE_{max}	**10,14**	12,33	**5,74**	9,14	**9,02**	11,02
10 × 3 × W	RE_{min}	**−0,30**	0,00	**−1,05**	0,00	**−1,63**	0,00
	RE_{avg}	**3,15**	4,68	**1,81**	3,61	**1,66**	3,31
	RE_{max}	**11,76**	16,07	**11,25**	14,71	**12,21**	14,66
20 × 2 × W	RE_{min}	**−0,19**	0,00	**−1,03**	0,00	**−0,14**	0,00
	RE_{avg}	**4,38**	4,76	**1,71**	2,05	4,10	**3,84**
	RE_{max}	**11,77**	11,81	9,22	**9,08**	**11,25**	12,44

As it could be seen in Table 10.6, the values of the considered types of the REs of the solutions found by the PLA3 for the considered problem sizes $n \times m$ and the discretisation levels W, in 23 out of 27 cases were lower than the values of the REs of the solutions found by AX-m version of the PLA2. The value of the RE that is lower than the RE of another considered algorithm is given in bold font. As it could be seen in Table 10.6, it's impossible to determine unequivocally the discretisation level W for which the values of the REs of the found solutions are always the lowest. However, REs yielded by both algorithms for $W = 20$ in a predominant number of cases are the lowest, thus the following relations between the REs can be formulated: $\text{REs}(W = 20) < \text{REs}(W = 50) < \text{REs}(W = 10)$. This might impose the conclusion, that the high discretisation level does not necessarily ensure the lowest values of the REs and the additional research is needed to identify the most appropriate discretisation of the continuous resource.

In addition, we give in Table 10.7 the percentage of the problem instances for which best solutions found by the PLA3 within 43 runs were better (3rd column) or not worse (4th column) than the best-known ones, specified for all of the considered discretisation levels. As a matter of fact, the third column shows the percentage of the problem instances for which the best-known solutions were improved by the PLA3 within 43 runs.

Finally, it should be mentioned, that within 43 runs, for combination 10 × 2, i.e. 10 tasks scheduled on 2 machines, the PLA3 was able to improve 33 out of 100 best known solutions, for combination 10 × 3 – 60 best known solutions, and for combination 20 × 2, the PLA3 improved 30 best known solutions. Altogether,

Table 10.7 The percentage of the problem instances for which best solutions found by the PLA3 within 43 runs were better (3rd column) or not worse (4th column) than the best-known ones specified for all of the considered discretisation levels

Problem size $n \times m \times W$	Discretisation level W	% of the improved the best-known solutions	% of not worse than the best-known solutions
$10 \times 2 \times W$	10	0	0
	20	9	22
	50	26	34
$10 \times 3 \times W$	10	1	2
	20	21	28
	50	48	54
$20 \times 2 \times W$	10	1	1
	20	28	30
	50	2	2

within 43 runs, the PLA3 improved 123 best known solutions, i.e. 41% of 300 instances of the considered problem. It should be also mentioned, that all the conclusions are valid for the particular implementation of the procedures used in the experiments. The values of some parameters of the learning procedures were determined during their tuning and should be verified on the way of exhaustive experiment.

10.5 IBDEA—Island-Based Differential Evolution Algorithm

In an island-based differential evolution algorithm (IBDEA), proposed in [25], two ideas were exploited, namely, the Differential Evolution method, first proposed in [22], and an island model, adopted for evolutionary computation, e.g. [1, 2, 21]. In the IBDEA, the evolutionary process is performed on an archipelago which consists of cooperating with each other autonomous islands. The population on an island consists of two halves of size x_{DE} each. The individuals in the first half—target vectors, are transformed, with help of mutation and crossover operators, into trial vectors which are placed in the second half of the population. The idea of keeping the offspring in the current population was borrowed from [26]. The whole evolutionary process is carried out using differential evolution algorithm (DEA), proposed in [24], which was adapted by the authors for solving DCSPwCRD. In the IBDEA, the islands cooperate with each other, cyclically sending their best solution

to one randomly chosen island. The process of evolution stops, when the predefined number of fitness function evaluations is carried out on the archipelago. The best across all islands individual is the final solution, found by the IBDEA to the considered problem. In the rest of the paper, we will use notions "an individual" and "a solution" interchangeably.

In the IBDEA, the assumptions made for the individuals are the same as the assumptions made for the individuals of the IBEA on the population level, see Sect. 10.1. The general description of the proposed IBDEA is given below.

```
Procedure IBDEA
Begin
  Set K ≥ 2 (K - the number of islands). Assign DEA pro-
  cedure to all islands.
  Assume the population of individuals P_k on an island k
  consists of two halves P^1_k and P^2_k, i.e. P_k = P^1_k + P^2_k,
  and |P_k| = x_P = 2·x_DE, |P^1_k| = x_DE, |P^2_k| = x_DE.
  Generate an initial population of individuals P^1_k of
  the size x_DE on every island k, k = 1, 2, … , K.
  Improve individuals on all islands with the DEA proce-
  dure, cyclically exchanging best individuals among
  randomly chosen pairs of islands.
  Stop after n_ev number of fitness function evaluations
  on the archipelago have been carried out.
  Output the best solution to the problem.
EndProc_IBDEA.
```

The individuals in the initial population are generated in such a way, that the position of a task in vector S, as well as the task's processing mode is chosen at random with the uniform distribution.

The solution exchange among the islands occurs cyclically, after $n_{ex} \ll n_{ev}$ number of the fitness function evaluations which have been carried out on every island. The pairs of islands, chosen at random from all the islands, exchange between themselves their best solutions. The random interconnection topology among islands was chosen as the most efficient according to [23].

The DEA procedure used in the IBDEA is described by the following pseudo code.

Procedure DEA

Begin

For each individual (a target vector S_{tg}) in population P^I_k do:

Choose at random from P^I_k three vectors: S_0, S_1, S_2.

Carry out the transformation stage:

create $S' = [S'(p_i) | i = 1, 2, \ldots, n]$ for each of S_{tg}, S_0, S_1, S_2, where S' is a tasks' positions vector and $S'(p_i)$ is the position of task i in S,

create modes vector $S'' = [S''(l_i) | i = 1, 2, \ldots, n]$ for each of S_{tg}, S_0, S_1, S_2, where $S''(l_i)$ is the mode of task i, in which task i is processed in S.

Carry out the mutation stage:

create $M' = [M'(p_i) | i = 1, 2, \ldots, n]$ - a tasks' positions mutant vector, calculating $M'(p_i)$ - the mutated position of task i as:

$$M'(p_i) = S'_0(p_i) + A \cdot rand \cdot (S'_1(p_i) - S'_2(p_i)), \quad (10.13)$$

create $M'' = [M''(l_i) | i = 1, 2, \ldots, n]$ - a modes' mutant vector, calculating $M''(l_i)$ - the mutated mode of task i as:

$$M''(l_i) = S''_0(l_i) + A \cdot rand \cdot (S''_1(l_i) - S''_2(l_i)). \quad (10.14)$$

Carry out the crossover stage:

create a task trial vector $T' = [T'(p_i) | i = 1, 2, \ldots, n]$, determining $T'(p_i)$ - the position of task i in T, as:

$$T'(p_i) = \begin{cases} M'(p_i) \text{ if } rand \le Cr_p \text{ or } i = rand(j) \\ S'_{tg}(p_i) \text{ if } rand > Cr_p \text{ and } i \ne rand(j), \end{cases} \quad (10.15)$$

create a mode trial vector $T'' = [T''(l_i) | i = 1, 2, \ldots, n]$, determining $T''(l_i)$ - the mode of task i in T, as:

$$T''(l_i) = \begin{cases} M''(l_i) \text{ if } rand \le Cr_l \text{ or } i = rand(j) \\ S''_{tg}(l_i) \text{ if } rand > Cr_l \text{ and } i \ne rand(j), \end{cases} \quad (10.16)$$

Carry out the repair stage:

order the tasks in T' according to the ascending values of $T'(p_i)$, this way the new repaired tasks' positions $T'_r(p_i)$ are determined,

create $T'_r = [T'_r(p_i) | i = 1, 2, \ldots, n]$ - a repaired tasks' positions vector,

create $T''_r = [T''_r(l_i) | i = 1, 2, \ldots, n]$ - a repaired tasks modes vector, where the repaired tasks modes $T''_r(l_i)$ are determined as follows:

$$T''_r(l_i) = \begin{cases} 1 \text{ if } T''(l_i) \le 1, \\ W \text{ if } T''(l_i) \ge W_i, \\ \lfloor T''(l_i) \rfloor \text{ if } 1 < T''(l_i) < W_i. \end{cases} \quad (10.17)$$

End_For

Combine T'_r and T''_r into $T = [c_i | \ 1 \le i \le n]$.

Evaluate T and save in P^2_k.

Select the best x_{DE} individuals from P_k in order to create the next generation of individuals.

Stop after the pre-set number of fitness function evaluations have been carried out.

EndProc_DEA.

A scale factor A, used in DEA procedure, controls the evolution rate of the population. The values of the variable $rand \in [0, 1]$. The crossover constants Cr_p and Cr_l control the probability, that the trial individual will receive the target individual's tasks positions or modes, where p and l in the notations Cr_p and Cr_l stand for tasks positions and modes respectively.

10.5.1 Computational Experiment

Proposed island-based differential evolution algorithm (IBDEA) for solving discrete-continuous scheduling problem with continuous resource discretisation Θ_Z was implemented and tested. There were 19 islands used to realize the IBDEA. The differential evolution algorithm (DEA), described in [24], has been adapted for solving considered Θ_Z and assigned to every island in the IBDEA. After preliminary tuning, the size of the population on every IBDEA island was set $x_{DE} = 200$, which is different from the sizes considered in [24], where $x_{DE} \in \{20, 40, 80, 60, 100\}$. The rest of the parameters necessary to carry out the differential evolution algorithm were set to the same values as in [24], namely the scale factor A, which controls the evolution rate of the population, was set $A = 1{,}5$ and the values of the variable $rand \in [0, 1]$. The crossover constants Cr_p and Cr_l which control the probability that trial individual will receive actual individual's tasks or modes were set $Cr_p = 0{,}2$ and $Cr_l = 0{,}1$, where p and l in the notations Cr_p and Cr_l stand for tasks positions and modes respectively. On every IBDEA island, an initial population of feasible individuals was generated using the uniform distribution equal $1/n$ for the tasks, and $1/W$ for the task's modes. For testing purposes three combinations of $n \times m$ were considered (n—the number of tasks and m—the number of machines): 10×2, 10×3, and 20×2. For each combination $n \times m$ 100 instances of a problem Θ_Z were generated and three discretisation levels W were considered: 10, 20, and 50. This way we considered nine sizes of the problem: $10 \times 2 \times 10$, $10 \times 2 \times 20$, $10 \times 2 \times 50$, $10 \times 3 \times 10$, …, $20 \times 2 \times 50$, which makes 900 instances of the problem in total. Each instance was tested 43 times. Mean time required by the IBDEA to find a solution for the problem sizes 10×2 and 10×3 for all discretisation levels on a PC under 64-bit operating system Windows 7 Enterprise with Intel(R) Core(TM) i5-2300 CPU @ 2.80 GHz 3.00 GHz, RAM 4 GB compiled with aid of Borland Turbo Delphi for Win32 was approximately 2−3 s, and for the problem size 20×2 for all discretisation levels approximately 5−6 s.

In order to evaluate the efficiency of the IBDEA, we have used three types of relative errors: minimum, average, and maximum relative error of the solutions yielded by the algorithm. Relative errors (REs) of the solutions compared to the best-known solutions were calculated according to the formulae: $RE = (Q_{algm} - Q_{best\text{-}known})/Q_{best\text{-}known}$, where Q_{algm}, $Q_{best\text{-}known}$—the schedule length of a solution found by the considered algorithm and the best-known solution respectively. The set of the best-known solutions was determined by the authors while using all designed by them algorithms and procedures for solving problem Θ_Z. We have determined RE_{min}, RE_{avg}, and RE_{max} for every size of the considered problem as a minimum, average, and maximum RE, respectively, across 4300 REs calculated, while solving each of the 100 instances 43 times. We have compared the REs of the solutions found by the IBDEA built on 19 islands to the REs of the solutions found by the IBDEA built on a single island, in other words the DEA itself. The values of such parameters as A, the variable $rand$, Cr_p and Cr_l were set to

the same values as in the IBDEA. We have also compared the REs of the solutions found by the IBDEA and the DEA to the REs of the solutions found by the PLA3 described in [21]. The values of RE_{min}, RE_{avg} and RE_{max} for the IBDEA, the DEA and the PLA3 for all problem sizes and considered discretisation levels are presented in Table 10.8. The smallest values of the respective REs for particular cases are given in bold font.

The values of REs in Table 10.8 show how much schedules yielded by the IBDEA were longer than the best known schedule for the same case. For example, for the case $10 \times 2 \times 10$ $RE_{avg} = 2{,}31\%$ means that the schedule length of all schedules yielded by the IBDEA was on average 2,31% longer than the best-known. For the same case, $RE_{max} = 7{,}90\%$ means that the longest schedule among all schedules yielded by the IBDEA was 7,90% longer than the best-known. For the case $10 \times 3 \times 10$ for the IBDEA and the PLA3, $RE_{min} = -0{,}3\%$ is negative, which means that the schedules found by the algorithms were shorter than the best-known for 0,3%. The algorithm whose REs values are the smallest is considered to be more efficient than the others. In Table 10.8, each of three considered algorithms is characterised by nine values of each type of REs for each considered problem size $n \times m \times W$. In 9 out of 9 cases, REs_{max} of the solutions, found by the IBDEA, were smaller than the REs_{max} of the DEA and the PLA3. The remaining REs_{avg} and REs_{min} of the IBDEA were the smallest in 6 and in 2 cases respectively. However, REs_{avg} and REs_{min} of the PLA3, in 3 and in 4 cases out of 9 were the smallest. In all cases, all types of the DEA's REs were the largest. As a general conclusion, we point out that in 17 cases out of 27, the IBDEA shows the smallest REs, and no other algorithm shows the same or better results. The PLA3 was the best in 8 cases out of 27, and the DEA only once achieved the same result as the IBDEA and the PLA3 (RE_{min} for the problem size $10 \times 2 \times 10$).

As it could be seen in Table 10.8, it's impossible to determine unequivocally the discretisation level W for which the values of the REs of the found solutions are always the smallest. However, REs yielded by all considered algorithms for $W = 20$ in a majority of cases were the smallest, thus the following relations between the REs can be formulated: $REs(W = 20) < REs(W = 50) < REs(W = 10)$. This might impose the conclusion, that high discretisation level does not necessarily ensure the smallest values of the REs and the additional research is needed to identify the most appropriate discretisation of the continuous resource.

The computational experiment shows, that the island model exploiting DE finds solutions whose relative errors (REs) are smaller than the REs of the solutions found by the DEA alone, see Table 10.8. The above statement is true under assumption, that on every island of the IBDEA operates the DEA, and that the size of the population on every island is the same as the size of the population in the DEA. In major number of cases, RE_{min} and RE_{avg} of the solutions found by the IBDEA were smaller than the RE_{min} and RE_{avg} of solutions found by the DEA and the PLA3, which also exploits the island model. The direct benefit of the proposed algorithm is its ability to find high quality solutions with a smaller dispersion of RE's values. In our experiment, in all cases, RE_{max} of the solutions found by the IBDEA were the smallest. The promising results achieved by the IBDEA might

Table 10.8 The comparison of the relative errors RE of solutions found by the IBDEA, the DEA, and the PLA3 for problem Θ_z

Problem size $n \times m \times W$	Relative error (RE)	Discretisation Level								
		$W = 10$			$W = 20$			$W = 50$		
		IBDEA (%)	DEA (%)	PLA3 (%)	IBDEA (%)	DEA (%)	PLA3 (%)	IBDEA (%)	DEA (%)	PLA3 (%)
$10 \times 2 \times W$	RE_{min}	**0,01**	**0,01**	**0,01**	−0,33	0,00	**−0,42**	**−0,65**	−0,53	−0,49
	RE_{avg}	**2,31**	3,86	2,70	**0,93**	2,86	1,53	**0,83**	3,22	1,81
	RE_{max}	**7,90**	11,52	10,14	**4,57**	10,72	5,74	**4,91**	11,86	9,02
$10 \times 3 \times W$	RE_{min}	**−0,30**	−0,10	**−0,30**	−0,75	−0,04	**−1,05**	**−1,71**	−1,12	−1,63
	RE_{avg}	**2,60**	4,29	3,15	**0,96**	3,18	1,81	**0,46**	3,02	1,66
	RE_{max}	**10,04**	13,31	11,76	**8,60**	17,16	11,25	**7,63**	16,80	12,21
$20 \times 2 \times W$	RE_{min}	0,78	0,33	**−0,19**	−0,78	−0,20	**−1,03**	1,29	−0,03	**−0,14**
	RE_{avg}	5,03	4,93	**4,38**	5,22	4,18	**1,71**	6,94	5,24	**4,10**
	RE_{max}	**10,11**	11,91	11,77	**9,20**	11,37	9,22	**10,53**	13,22	11,25

suggest its superiority over the DEA, however in order to make the final conclusion more extensive research is needed.

Finally, it should be also mentioned, that all the conclusions are valid for the particular implementation of the procedures used in the experiments. The values of some parameters of the learning procedures were determined during their preliminary tuning and should be verified on the way of the exhaustive experiment.

References

1. Alba, E., Troya, J.: Analysis of synchronous and asynchronous parallel distributed genetic algorithms with structured and panmictic islands. In: Rolim, J. et al., (eds.) Proceedings of the 10th Symposium on Parallel and Distributed Processing, pp. 248–256. San Juan, Puerto Rico, USA, 12–16 April (1999)
2. Belding, T.C.: The distributed genetic algorithm revisited. In: Eshelman, L.J. (ed.) Proceedings of the Sixth International Conference on Genetic Algorithms, pp. 114–121. Morgan Kaufmann, San Francisco CA (1995)
3. Gordon, V.S., Whitley, D.: Serial and parallel genetic algorithms as function optimizers. In: Forrest, S. (ed.) Proceedings of the Fifth International Conference on Genetic Algorithms, pp. 177–183. Morgan Kaufmann, San Mateo, CA (1993)
4. Czarnowski, I., Gutjahr, W.J., Jędrzejowicz, P., Ratajczak, E., Skakowski, A., Wierzbowska, I.: Scheduling multiprocessor tasks in presence of correlated failures. In: Luptaćik, M., Wildburger, U.L. (eds.) Central European Journal of Operations Research, vol. 11, iss. 2, pp. 163–182. Physika-Verlag, Springer, Heidelberg (2003)
5. Jędrzejowicz, P., Skakovski, A., Czarnowski, I., Szreder, H.: Evolution-based scheduling of multiple variant and multiple processor programs. In: Hertzberger, L.O., Sloot P.M.A. (eds.) Future Generation Computer Systems, vol. 17, pp. 405–414. Elsevier, The Netherlands (2001)
6. Jędrzejowicz, P., Skakovski, A.: An island-based evolution algorithm for discrete-continuous scheduling with continuous resource discretisation. In: Proceedings of the 2nd IEEE International Conference on Computational Cybernetics ICCC 2004, 30 Aug–1 Sep 2004. Vienna University of Technology, Austria (2004)
7. Różycki, R.: Zastosowanie algorytmu genetycznego do rozwiązywania dyskretno-ciągłych problemów szeregowania. PhD Dissertation, Poznań University of Technology, Poland (2000)
8. Jędrzejowicz, P., Skakovski, A.: A population learning algorithm for discrete-continuous scheduling with continuous resource discretisation. In: Chen, Y., Abraham, A. (eds.) Proceedings of 6th International Conference on Intelligent Systems Design and Applications (ISDA 2006), vol. 2, spec. sess.: Nature Imitation Methods Theory and practice (NIM' 06), pp. 1153–1158. Jinan, Peoples R. of China (2006)
9. Jędrzejowicz, P.: Social learning algorithm as a tool for solving some difficult scheduling problems. Found. Comput. Decis. Sci. **24**, 51–66 (1999)
10. Glover, F.: Tabu search: a tutorial. Interfaces **20**, 74–94 (1990)
11. Jędrzejowicz, P., Skakovski, A.: A cross-entropy based population learning algorithm for discrete-continuous scheduling with continuous resource discretisation. Neurocomputing **73** (4–6), Special Issue: SI:655–660 (2010)
12. Rubinstein, R.Y.: Optimization of computer simulation models with rare events. Eur. J. Op. Res. **99**, 89–112 (1997)
13. De Boer, P.-T., Kroese, D.P., Mannor, S., Rubinstein, R.Y.: A tutorial on the cross-entropy method. Ann. Op. Res. **134**(1), 19–67 (2005)

14. Cantu-Paz, E., Goldberg, D.E.: Are multiple runs of genetic algorithms better than one?. In: Proceedings of the Genetic and Evolutionary Computation Conference (2003)
15. Whitley, D., Rana, S., Heckendorn, R.B.: The island model genetic algorithm: on separability, population size and convergence. J. Comput. Inf. Technol. **7**(1), 33–47 (1999)
16. Cantu-Paz, E.: Migration policies, selection pressure, and parallel evolutionary algorithms. J. Heuristics **7**(4), 31–334 (2001)
17. Skolicki, Z., Kenneth, D.J.: The influence of migration sizes and intervals on island models. In: Proceedings of GECCO' 05, pp. 1295–1302. Washington, DC, USA, 25–29 June (2005)
18. Krink, T., Mayoh, B.H., Michalewicz, Z.: A PACHWORK model for evolutionary algorithms with structured and variable size populations. In: Morgan, K., Banzhaf, W., Daida, J., Eiben, A.E., Garzon, M.H., Honavar, V., Jakiela, M., Smith, R.E. (eds.) Proceedings of the Genetic and Evolutionary Computation Conference, vol. 2, pp. 1321–1328. Orlando, Florida, USA (1999)
19. Sekaj, I.: Robust parallel genetic algorithms with re-initialisation. In: Proceedings of Parallel Problem Solving from Nature—PPSN VIII, 8th International Conference, vol. 3242, pp. 411–419. LNCS, Springer, Birmingham, UK, 18–22 Sep (2004)
20. Skolicki, Z.: An analysis of island models in evolutionary computation. In: Proceedings of GECCO' 05, pp. 386–389. Washington, DC, USA, 25–29 June (2005)
21. Jędrzejowicz, P., Skakovski, A.: Population learning with differential evolution for the discrete-continuous scheduling with continuous resource discretisation. In: IEEE International Conference on Cybernetics (CYBCONF) pp. 92–97. Lausanne, Switzerland, 13–15 June (2013)
22. Storn, R., Price, K.: Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces. J. Global Opt. **11**, 341–359 (1997)
23. Jędrzejowicz, P., Skakovski, A.: Structure vs. efficiency of the cross-entropy based population learning algorithm for discrete-continuous scheduling with continuous resource discretisation. In: Czarnowski, I., Jędrzejowicz, P., Kacprzyk, J. (eds.) Studies in Computational Intelligence. Agent-Based Optimization, vol. 456, pp. 77–102 (2013)
24. Damak, N., Jarboui, B., Siarry, P., Loukil, T.: Differential evolution for solving multi-mode resource-constrained project scheduling problems. Comput. Op. Res. **36**(9), 2653–2659 (2009)
25. Jędrzejowicz, P., Skakovski, A.: Island-based differential evolution algorithm for the discrete-continuous scheduling with continuous resource discretisation. Procedia Comput. Sci. **35**, 111–117 (2014)
26. Kazemipoor, H., Tavakkoli-Moghaddam, R., Shahnazari-Shahrezaei, P.: Differential evolution and simulated annealing algorithms for a multi-skilled project scheduling problem. Am. J. Sci. Res. **33**, 136–146 (2011)