INTL JOURNAL OF ELECTRONICS AND TELECOMMUNICATIONS, 2012, VOL. 58, NO. 4, PP. 369–379
Manuscript received October 31, 2012; revised Decemer, 2012. DOI: 10.2478/v10177-012-0051-1

Problem-Independent Approach to Multiprocessor Dependent Task Scheduling

Dawid Król, Dawid Zydek, and Leszek Koszałka

***Abstract*—This paper concerns Directed Acyclic Graph task scheduling on parallel executors. The problem is solved using two new implementations of Tabu Search and genetic algorithm presented in the paper. A new approach to solution coding is also introduced and implemented in both metaheuristics algorithms. Results given by the algorithms are compared to those generated by greedy LPT and SS-FF algorithms; and HAR algorithm. The analysis of the obtained results of multistage simulation experiments confirms the conclusion that the proposed and implemented algorithms are characterized by very good performance and characteristics.**

***Keywords*—Tasks scheduling, DAG, genetic algorithm, Tabu Search, makespan.**

I. Introduction

TASK SCHEDULING is one of the oldest and the most important problem in computer science. In the most general version of this problem, there is given a set of tasks and system containing some number of processors (executors). The goal of scheduling is to assign tasks to executors in such a way that specified criterion is minimal or maximal. Although scheduling problem is specific for computer science, it can be easily found in daily life, industry, and business [1]. Advanced multi-criteria scheduler is implemented in every Enterprise Resource Planning system. In these systems, scheduler is used in Master Requirements Planning, Production Activity Control or Supplies Chain Management. Recently, scheduling problem became very important issue in distributed computing systems. These systems are one of the fastest growing areas in computer science. Examples of these systems are those created using software like Globus Toolkit, Legion or BOINC [2]. Many problems in the category of task scheduling problems are NP-hard, that unflagging interest in this area.

Basic and the simplest scheduling problem is to assign independent and unrelated tasks to processors. System contains known and constant number of identical processors. Each processor is continuously available and can process one task at a time. Task duration is known. Each task has to be executed by exactly one processor. Set of tasks is given and it is not changed in time [3]–[5]. Release time of task is zero so each task can be started in any moment. Tasks are unrelated and independent; therefore there are no precedence restrictions. Execution of any task does not imply execution of other tasks thus each task can be executed without considering completion of other tasks before. There is no deadline for any task. Execution of a task cannot be stopped and when task is started it has to run till completion. The basic scheduling problem described above was considered in [6]–[9]. There are three features of scheduling problems dictating the specific scheduling problem: i) Type of executors; ii) Tasks; and iii) Evaluation criterion. Huge diversity of scheduling problems results in many research projects and scientific papers.

In practical implementations, tasks in scheduling problem are very often related to each other. A great example is process of production, where first task describes preparing parts and second assembling them. Assembling can be performed only if all parts were created earlier. This problem was created by using order dependences to the basic scheduling problem described above. In literature set of related tasks is presented as a Directed Acyclic Graph (DAG scheduling problem). This issue was studied in [10]–[12].

Another instance of scheduling problem can be obtained by allowing assignment of one task to more than one processor [13], [14]. It results in decreasing duration of task processing. This kind of scheduling problems is especially important nowadays due to expansion of General-purpose Computing on Graphic Processor Unit (GPGPU). GPGPU idea is to execute tasks in parallel using multi cores.

Computing in distributed systems is also growing area nowadays. These systems contain significant number of computers connected by a network. Nodes of distributed system are usually spread across some area. Due to significant delay in communication among processing elements, executing one task on many processors is used to guarantee its execution rather than speeding it up. Emphasis is moved here to taking into account these delays. The problem is called DAG scheduling with communication delay [15]–[24].

Significant variation of the basic scheduling problem was developed in factories. Production process very often demands using different machines. Machines are dedicated to perform some strictly specified activities and they are unable to perform others. Great example is welding and painting machine. Implementation of dedicated machines enforces significant changes in the scheduling problem. In general, task becomes an item that has to visit all machines; and set of tasks contains many items. The scheduler is responsible in this case for assigning tasks to each machine. This problem was considered in [25]–[28].

In this paper, basic scheduling problem with precedence limitation is considered. Two new algorithms namely TrAl-TS (Tabu Search) and TrAl-GA (Genetic Algorithm) are proposed. Detailed examination and analysis of advantages and

D. Król and D. Zydek are with the Department of Electrical Engineering, Idaho State University, USA (e-mails: kroldawi@isu.edu; zydedawi@isu.edu).
L. Koszałka is with the Department of Systems and Computer Networks, Wrocław University of Technology, Poland (e-mail: leszek.koszalka@pwr.wroc.pl).

disadvantages of the presented algorithms and other algorithms [7], [19] is conducted and described. Proposed in this paper algorithms are also compared to the well-known metaheuristic and greedy methods.

The rest of the paper is composed as follows: In Section II, the mathematical model of the problem is presented. Section III contains description of proposed algorithms and method of solution coding. Section IV contains a design of experiment. In Section V, the results of investigations are presented, followed by discussion. Section VI contains final remarks.

II. Problem Statement and Nomenclature

The mathematical model of the considered task scheduling problem is given in three-field notation as $P|precc|C_{max}$, where first part defines executors, second describes set of tasks, and third specifies evaluation criterion [1].

System $P = \{1, 2, ..., p\}$ consists of p parallel and identical processors. Parallel processors means, that all of them can execute assigned tasks independently and simultaneously. Identical processors implies equation (1).

$$l_{i1} = l_{i2} = ... = l_{ip} = l_i \tag{1}$$

, where i is a considered task, l_{i1} is time that is needed to execute the task i on 1st processor, and finally l_i denotes execution time in which every processor would complete the task i. This means that execution time of specified task is the same for every processor in the system. In addition, it is assumed that processors are connected with each other, but the time needed for communication among them is negligibly small and it is therefore omitted.

The tasks to be scheduled are given as DAG. Figure 1 contains an example of DAG. The graph is described by $G = \{T, E, L\}$, where $T = \{1, 2, ..., n\}$ is set of tasks, $E = \{e_{ij} | i, j \in T\}$ is set of edges that represent precedence among tasks, and $L = \{l_1, l_2, ..., l_n\}$ is set of tasks' duration. A task denoted by i cannot be executed until all of its predecessors are not completed.

In this paper, we assume the following assumptions:

- all tasks are indivisible;
- one task cannot be executed by more than one processor;
- there are no other precedence despite given set E;
- all tasks have the same arrival time that is equal to 0;
- there is no deadline for the tasks.

In this paper, the following notations are used to discuss problem:

- for an edge $e_{ij} \in E$, task i is called predecessor of task j and task j is successor of task i;
- $PRED(i)$: set of all predecessors of task i;
- $SUCC(i)$: set of all successors of task i;
- $t(i)$: moment when execution of task i is begun;
- $f(i)$: moment when task i is completed;
- $w(k)$: time when all tasks assigned to processor k are completed;
- $x(i)$: identifies execution of task i by a processor.

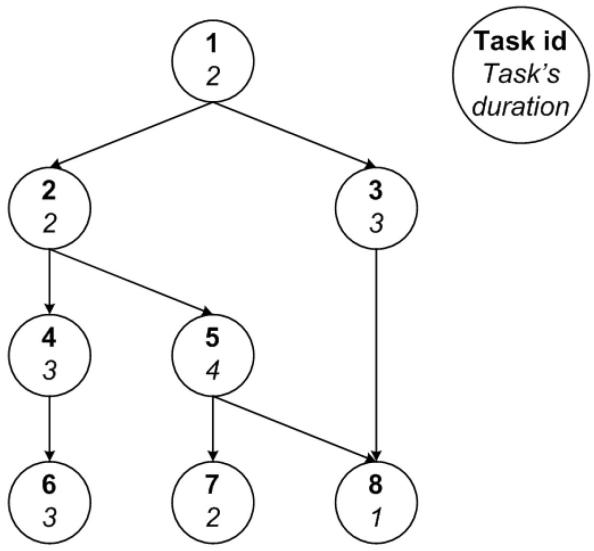

Fig. 1. Example of DAG

To evaluate solutions, a schedule length criterion is used. It is also called makespan criterion and it specifies the completion time of all tasks. The completion time is affected by both task's duration and completion time of all its predecessors. The evaluation criterion is equal to the moment when all tasks are completed, and it is expressed by:

$$C_{max} = max\{f(1), f(2), ..., f(n)\} \tag{2}$$

Solving considered problem means to assign all tasks to the processors. Solution of the problem is called schedule $H = \{S, L, X, P\}$, where $S = \{t(1), t(2), ..., t(n)\}$ is a set of moments when tasks started to be executed, L is set of tasks' durations, $X = \{x(1), x(2), ..., x(n)\}$ includes elements identifying, which processor is assigned to each task; and P is a set of executors. Figure 2 contains example of schedule.

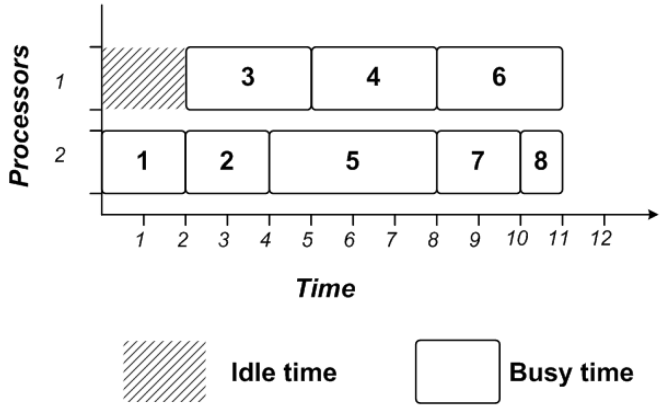

Fig. 2. Sample schedule.

The schedule is considered to be feasible if all tasks from a given set of tasks are scheduled, precedence is satisfied, and all assumptions are met. The objective is to minimize the maximum task completion time that is denoted by C_{max}.

III. Metaheuristic Algorithms

We propose two different methods of solving the scheduling problem using metaheuristic algorithms. First of them is Tabu Search algorithm and second is genetic algorithm. Although both algorithms were widely used in many other papers, there is a factor that distinguishes considered approaches from other solutions. Both of them employ a new method of coding and evaluating solution. Our intention was to improve solutions generated by other algorithms and to make implementation of algorithms more flexible. This section contains description of the proposed coding method and method of solution evaluation along with their motivations. The designed algorithms are also described in this section.

A. Solution Coding

1) Motivation: As it is commonly known in the literature, the most important and the most difficult problem in the design of the metaheuristic algorithms is to determine the method of results coding and evaluation function.

Encoding the solution is to express the solution in a form that is understandable for the algorithm and allows performing operations specific to the metaheuristic algorithm. In the literature, there are distinguished three categories of coding. First of them is a many-to-one, where many different solutions can be encoded in the same way. The second possibility is one-to-many assuming that the same solution can be encoded in many different ways. The last category is called one-to-one. This way of coding ensures that one result will be encoded unambiguously. From a design perspective, one of the best technique is the one-to-one, because it greatly simplifies the design of evaluation function and metaheuristic operations.

The evaluation function is the assignment that allots a value to solution. This value characterizes solution and represents its value. Result of evaluation function is used to compare the solutions and to indicate the better one. Evaluation of solutions is the basis for each metaheuristic algorithm. To meet its role, assignment has to be unambiguous. This means that for every solution, one and only one value can be assigned.

Both, coding method and evaluation function, can strongly influence metaheuristic operators. For this reason most of the implementations of metaheuristic algorithms cannot be used (without modification) to solve other optimization problems.

In problem considered in this paper, coding function must take into account not only the executor, to which task is assigned, but also the moment in which execution of this task is started. In addition, the encoding and evaluation function cannot ignore precedence among tasks. In order to avoid designing inflexible and complicated metaheuristic operators, new approach to problem notation, called TrAl, has been proposed. The idea is to unify the representation of a solution in such a way that it corresponds with any optimization problem. In addition, new notation method has to allow using every metaheuristic algorithm to solve any different optimization problem, without need of modifications of the metaheuristic operators. Solution S is given as $S = (A, R)$, where A is one-dimensional array of values, and R is a translation algorithm that converts an array A into human-readable and evaluable form.

To meet its role, the array A has to satisfy the following conditions:

- array A has to be one-dimensional in order to simplify implementation of metaheuristic operations;
- values stored in the array have to allow unambiguous translation to feasible solution;
- at the level of the metaheuristic algorithm the values in array are not related to the particular problem in any way;
- type of values stored in the array cannot affect the implementation of metaheuristic operations;
- values stored in the array can be given by any type that is dictated by the requirements of the algorithm.

The translation algorithm R also has to satisfy certain conditions:

- the algorithm uses only array A and the properties of the problem;
- the algorithm cannot modify values in array A;
- the algorithm translates array A to human-readable solution unambiguously;
- the algorithm calculates value of the evaluation function that is specified in problem model;
- the algorithm can have any structure and implementation;
- the algorithm takes as an input array A;
- output of the algorithm have to be a valid schedule that is understandable by a human and possible to assess.

The proposed approach provides a wide range of applications and new opportunities. Solutions can be used both as greedy algorithms and metaheuristics. Changing the optimization problem does not result in the necessity to modify the metaheuristic operators. It is enough to modify the translation algorithm R to a new problem. Translation algorithm can be implemented in many different ways. In the simplest cases it is responsible only for calculating the value of evaluation function. In more complex problems, translation algorithm can be even a separate heuristic algorithm that solves a subproblem.

2) Implementation: The presented approach was adapted to tasks scheduling problem. Array $A = \{a_1, a_2, ..., a_n | 1, 2, ..., n \in T\}$ consists of n elements, where $n \in T$. Each element from set A represents separate task defined by the index. Value of element is a priority of the task. The higher priority[1], the task has to be completed faster. Algorithm R is a method that assigns tasks to processors. Algorithm takes into account precedence. In each step, feasible task i with the highest priority is selected and assigned to specified executor. Selection of the processor k depends on execution time of the processor and earliest start time $t(i)$ of selected task. Executors with w smaller than $t(i)$ are preferable. Otherwise the executor with smallest w is selected. Algorithm 1 presents translation algorithm used in

[1]If set T contains n tasks, then the highest priority is 1 and the lowest is n.

this paper and Example 1 shows step-by-step performance of algorithm.

Algorithm 1: Translation algorithm R

1. **For** $i = 1$ **to** n
2. Create set FT that contains tasks feasible to be executed
3. Select form FT task i with the highest priority
4. Create $PREC(i)$
5. Determine the earliest moment $t(i)$ that task i can be started; $t(i) = max_{j \in PREC(i)}(f(j))$
6. Select executor $k : w(k) \leq t(i)$ and $w(k) - t(i)$ is minimal
7. **If** executor was found **Then**
8. Assign task i to executor k
9. Set moment of processing completion $f(i) = t(i) + l_i$; $w(k) = f(i)$
10. **Else**
11. Select executor k with the earliest moment of processing completion
12. Assign task i to executor k
13. Set task i execution start and stop moments $t(i) = w(k)$; $f(i) = t(i) + l_i$
14. Set $w(k) = w(k) + l_i$
15. **End If**
16. **End For**

Example 1: Performance of translation algorithm

Set of tasks given by DAG is presented in Fig. 3. Tasks will be scheduled on dual-processor system.

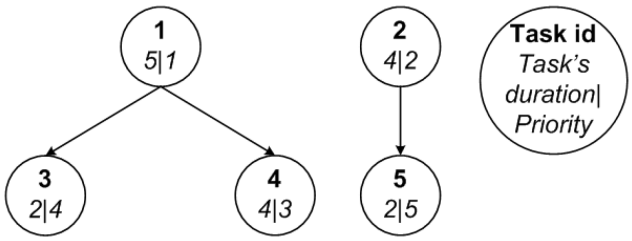

Fig. 3. Set of tasks to be scheduled.

Step 1. The feasible tasks are 1 and 2. Because the task 1 has higher priority than task 2, it will be scheduled as first. Set $PREC(1)$ is empty so the earliest moment of execution $t(1)$ of task 1 is equal to 0. Processing times of the two executors $w(1)$ and $w(2)$ are equal to 0, so the task 1 will be assigned to executor 1.

Step 2. The feasible tasks are 2, 3, and 4. Task with the highest priority is the task 2. $PREC(2) = \emptyset$ so the earliest $t(2)$ is 0. Because $w(2) = 0$, task 2 will be assigned to executor 2.

Step 3. There are three feasible tasks 3, 4, and 5. Task 4 should be scheduled as first because of higher priority. The earliest $t(4)$ is equal to the moment of completion all of its predecessors, which is $t(4) = max_{i \in PREC(4)}(f(i)) = 5$. The task 4 will be assigned to processor 1 because:

- $w(1) \leq (4)$;
- $t(4) - w(1) < t(4) - w(2)$.

Step 4. There are tasks 4 and 5 left. Due to higher priority, task 3 will be scheduled first. The earliest $t(3) = max_{i \in PREC(3)}(f(i)) = 5$. Task will be assigned to processor 2 because this is the only executor where $w(2) \leq t(3)$. Because task 3 cannot begin until all predecessors are not completed, the processor 2 has to wait idly for 1 unit of time.

Step 5. The last task is task 5. $PREC(5) = 2$ so the earliest $t(5) = max_{i \in PREC(5)}(f(i)) = 4$. However, $w(1) > t(5)$ and $w(2) > t(5)$, therefore task 5 is assigned to executor with the smallest processing time, which is executor 2.

The resulting schedule is presented in Fig. 4.

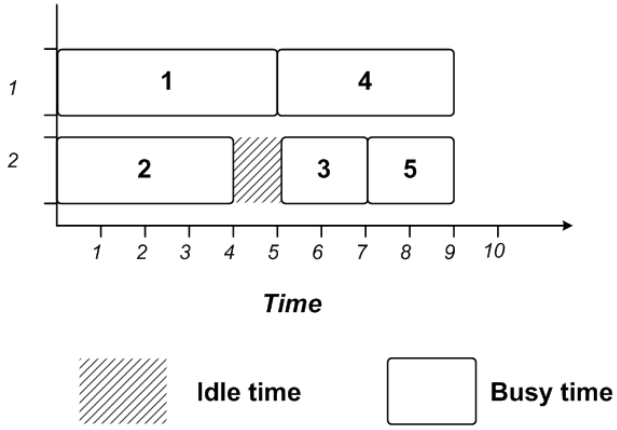

Fig. 4. Schedule generated from DAG considered in Example 1.

B. Tabu Search

Idea of Tabu Search algorithm was created in 1986 by Fred Glover [29]. His approach was motivated by basic disadvantages of local search. In each step of local search algorithm the best solution s' from neighborhood $N(s)$ of current solution s is selected. This strategy allows obtaining only the local optima. Therefore Fred Glover suggested some modifications which allow finding globally optimal solution.

First of the modifications is regard strategy of selecting best solution from neighborhood. To diversify and extend search space, Tabu Search permits two types of best solutions, which are globally-best and locally-best. These solutions will be called respectively gs and s. Solution gs is the best solution found among iterations and solution s is the best solution in interation. Solution s is changed in each iteration. This technique allows storing the best known solutions through iterations, escaping from local optima, and greatly increasing the search space [29]. Although technique presented above significantly extends search space and allows escaping from local optima, it also causes a new problem. Changing s in each iteration may cause cycles and as a result, it prevents further searching for optimal solution. To avoid cycles and prevent algorithm from stuck in dead end, a second mechanism, called tabu list, was suggested. Tabu list is a short term memory storing solutions that were used as the locally-best in previous

iterations. Each solution in tabu list cannot be selected as an s even if in fact, it is the best solution in the neighborhood. Solutions are removed from tabu list after some specified number of iterations [29].

Classic Tabu Search algorithm contains five steps. First, initial solution s is created. Second, neighborhood $N(s)$ is created. In third step, the best solution s' is selected and if it is better that the best found solution (gs), then s' is accepted as a new gs. In fourth step, the best solution s' that is not in tabu list is selected and accepted as new s. In fifth step, tabu list is updated. Steps from 2 to 5 are repeated until stopping criterion is met.

Tabu Search algorithm designed in this paper was called TrAl-TS and is described by Algorithm 2.

Algorithm 2: TrAl-TS algorithm

1. Create an initial solution s
2. Set $gs = s$
3. Create empty tabu list TL and frequency list FL
4. **For** $i = 1$ **to** $number\_of\_iterations$
5. Create neighborhood $N(s)$
6. Select the best s' from $N(s)$
7. **If** $f(s') \leq f(gs)$ **Then**
8. Set $gs = s'$
9. **End If**
10. Select the best and not prohibited s'
11. Set $s = s'$
12. Add move that was used to create s' to TL
13. Delete move that spent $tabu\_size$ iterations in TL
14. Update FL with move that was used to create s'
15. **If** gs was not changed for $unchanged$ number of iterations **Then**
16. Set s to solution generated with the least frequent used move from FL
17. **End If**
18. **End For**
19. **Return** gs

Important properties of TrAl-TS algorithm are as follows:

- parameters of TrAl-TS:
  - number of iterations ($number\_of\_iterations$);
  - tabu list size ($tabu\_size$) specifying how many solutions can be stored in the list. Since in each iteration one solution is added to tabu list, tabu list size is equal to the number of iterations spent by solution in tabu list;
  - number of iterations where gs can stay unchanged ($unchanged$) specifies, how many iterations gs can be unimproved until algorithm reacts;
- tabu list stores moves that were used to create solutions, instead of the solutions themselves. This approach significantly reduces memory requirements and allows the algorithm to run faster;
- long term memory is a histogram that shows how many times each possible move was used to generate a locally-best solution. After a specified by parameter $unchanged$ number of iterations without improvement of the globally-best solution, the algorithm changes s. Algorithm sets s to the solution generated with the least used solution from long term memory. This allows to make search space wider;
- neighborhood is generated using swap move. Figure 5 presents an example where that method is used.

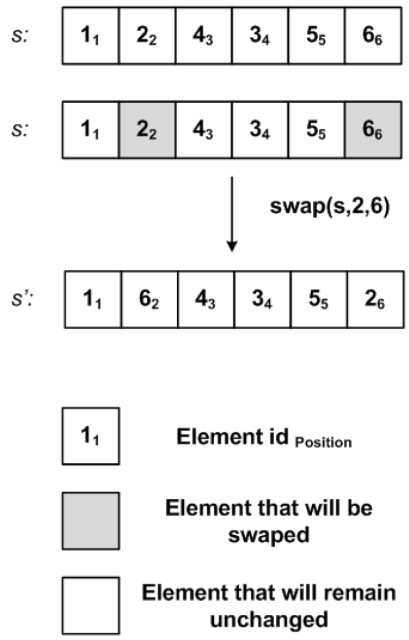

Fig. 5. Swap method.

C. Genetic Algorithm

Genetic algorithm takes its concept from process of natural evolution. Scientific researches in 1950 and 1960 in cellular anatomy field built the foundation of using evolution process to solve optimization problems. Implementation of this idea has begun in 1980. It was assumed that single solution of optimization problem is an individual or chromosome, which is created from genes.

At the beginning, initial population of individuals is created. In the environment where individuals live, there is a process of natural selection. Adaptation degree of individual is described by criterion function. During the selection, weak individuals are removed from population while the reaming group contains only the best chromosomes. This group takes a part in crossover process. During crossover process, two individuals are selected and they become parents for two new chromosomes. Offspring individuals contain only genes that were inherited from parents. Right after the procreation, mutation process starts. In this process a random change in some genes appears. After mutation, new individual is added to offspring population. Process of crossover and mutation is continued until new population reaches the specified size. This is the end of iteration (that is also called epoch). In

next epoch, previous offspring population becomes parent population and the process starts from the beginning. After completing specified number of iterations, the best individual from last offspring population is selected and returned as a result of algorithm [29].

Genetic algorithm considered in this paper is defined by Algorithm 3 and is called TrAl-GA.

Algorithm 3: TrAl-GA algorithm

1. Create initial parents population using random algorithm
2. **For** $i = 1$ **to** $number\_of\_iterations$
3. Create ranking of individuals from parents population
4. Add top $number\_of\_elite$ individuals to offspring population
5. **For** $j = number\_of\_best + 1$ **to** $population\_size$
6. Select two individuals from parents population using roulette wheel method
7. Perform crossover using PMX method
8. Perform mutation using Swap method
9. Add obtained individuals to offspring population
10. **End If**
11. Replace parents population with offspring population
12. **End If**
13. **Return** the best individual from parents population

Important properties of TrAl-GA algorithm are as follows::

- parameters of TrAl-GA:
  - number of iterations ($number\_of\_iterations$);
  - population size ($population\_size$) defines how many individuals are contained in the population;
  - best results ($number\_of\_best$) indicates how many the best individuals from population will survive process of natural selection;
  - elite results ($number\_of\_elite$) indicates how many the best individuals from population will be added to offspring population without any changes;
  - mutation probability ($mutation$) defines the probability that gene will randomly change its value;
- crossover operator used in the algorithm is one of the most commonly used operators in cases, where the chromosome is a permutation [29]. It is called Partially-Mapped Crossover (PMX). In the PMX method, two different split points are selected. These points indicate where the chromosomes will be divided. Initially, the first child is composed of the same genes as the first parent. Then, the first gene in the offspring after the first split point changes its value to the value of a gene located on the same position in the second parent. At the end, the child is searched for repetition of the new value. If repetition is found, the gene on this position is changed to previous value of gene considered in this iteration. The same process is performed for all genes between split points [29]. To clarify, the process of PMX crossover was presented in Fig. 6;
- mutation operator is defined exactly as neighbor generator for TrAl-TS. The main reason why the swap method is used instead of classic mutation operator is that solution is coded as a sequence of numbers. This sequence is a permutation. Therefore, classic random change of selected gene may cause infeasible schedule;

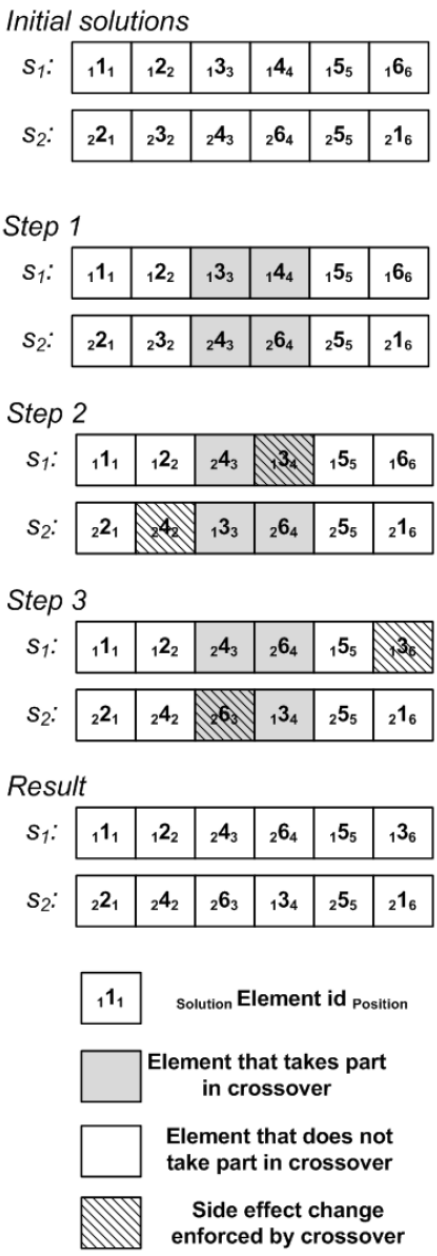

Fig. 6. PMX method.

- in the selection process, a modified method of the roulette wheel is used. The modification is based on the fact that the probability of selecting individual is not proportional to the value of evaluation function [29], but to the number of points assigned to the individual. Number of points depends on position that specified individual occupies in a ranking. The ranking consists of a number of places equal to the number of individuals involved in the selection. The ranking solutions are sorted by the value of evaluation function. The worst individual is placed in the last place in the ranking and scores 1 point. Best

individual ranks the first place, where number of points is equal to the number of individuals involved in the selection. Probability of choosing a solution, and hence the width of the field on the roulette wheel, is equal to the number of points, that the individual has, divided by the total number of points separated in the rankings;
- TrAl-GA algorithm uses elite mechanism. It means that specified number of the best solutions from the population will be added automatically to offspring population before the crossover process. Saving the best individuals in each epoch guarantees that quality of solutions will not decrease. The best results will survive through iterations and in worst case every next population will be at least same good as previous [29].

IV. Experimentations

As it is mentioned in previous section, two metaheuristic algorithms are proposed. In order to examine efficiency of these algorithms, obtained results have to be compared to some reference point. In this paper, the reference point is generated by metaheuristic algorithm HAR [19] and two greedy algorithms: Longest Processing Time (LPT) and Split Solution-First Fit (SS-FF). In addition, all results can be referred to optimal solutions, which are known for the selected problems.

Experiments consist of two parts:

- estimation of the best parameters values – in this part, all metaheuristic algorithms solve four test bench problem instances with different values of parameters. Results are gathered and analyzed. Values of parameters, where algorithms allow to generate the best results, are used in second part of the experiment;
- effectiveness analysis of proposed algorithms – in this part, all algorithms will be solving 16 instances of described scheduling problem. For all of them, optimal solution is known. Method of generating non-trivial DAG problems is described in next section. Specification of the selected problems is contained in Tab. I.

TABLE I
Problems Solved in the Experiments

ID	Number of processors	Number of tasks	Optimal solution
1	5	30	25
2	5	40	37
3	5	50	41
4	5	60	49
5	5	70	59
6	5	80	69
7	5	90	77
8	5	100	83
9	10	30	14
10	10	40	19
11	10	50	24
12	10	60	28
13	10	70	30
14	10	80	35
15	10	90	38
16	10	100	45

A. Generating Non-Trivial DAG Problems

The most important aspect of analyzing results of new algorithm is comparing them to some irrelative reference point. It can be obtained by using benchmarks or problems that were solved by well-known algorithms. However, the best reference point is an optimal solution. The main disadvantage of this approach is that for NP-hard problems optimal polynomial time algorithms do not exist and complete review of whole solution space is not an option. Nevertheless, it is possible to generate problem with known optimal solution.

In this paper, algorithm that generates non-trivial problem instances with known optimal solution was designed and used. The algorithm is presented below.

Algorithm 4: Problem generator

1. Specify n, p, $max(l(tasks))$, $min(l(task))$
2. Specify C_{max}
3. **For** $j = 1$ **to** p
4. **If** $j = 1$ **Then**
5. Assign to processor j m one-unit tasks where $m = C_{max}$
6. **Else**
7. Assign to processor j m one-unit tasks where $C_{max} - min(l(tasks)) \leq m \leq C_{max}$
8. **End If**
9. **End For**
10. Merge tasks with respect to $max(l(tasks))$ and $min(l(task))$ until n is met
11. Index tasks
12. Assign precedence among tasks with maintaining feasibility of schedule
13. **Return** DAG

B. Experimentation Methodology

Heuristic algorithms used in the experiment are strongly influenced by randomness. Initial solution is created with random algorithm, individuals in crossover operation are selected randomly, and crossover points are also random. Widely used randomness may result in different solutions obtained by an algorithm solving the same problem with the same parameters. In order to eliminate the risk of unreliable results, there is a need to repeat experiment for specified number of times, and then aggregate results.

In every experiment using TrAl-GA, TrAl-TS, or HAR algorithms, minimal number of repetitions is 10; and maximal number is 1000. However, usually number of repetitions does not reach upper bound. When new solution differs from median solution by no more than 10%, experiment is stopped. If so, then this solution is considered to be result of the single experiment. Algorithm estimating experiment solution is given by Algorithm 5.

Algorithm 5: Metaheuristic algorithms solution estimator

1. **For** $i = 1$ **to** 10
2. Execute algorithm for specified problem and set of parameters
3. **End For**
4. **While** value of evaluation function in new solution differs from median value obtained from evaluate function of previously obtained results by more than 10% **Do**
5. Execute algorithm for specified problem and parameters' set
6. **If** $i > 1000$ **Then**
7. **Exit Do**
8. **End If**
9. $i = i + 1$
10. **End Do**
11. **Return** newest solution

V. Results Analysis

Effectiveness of the algorithms is examined in three ways. First, effectiveness is expressed with respect to number of iterations. This allows checking the influence of increased number of iterations on quality of obtained results. Second, effectiveness is presented with respect to problem size. This shows how problem size affects performance of algorithms. Third analysis is general comparison of results obtained by examined algorithms.

A. Effectiveness with Respect to Number of Iterations

Figure 7 contains graph that illustrates average difference between results generated by metaheuristic algorithms and optimal results for different number of iterations.

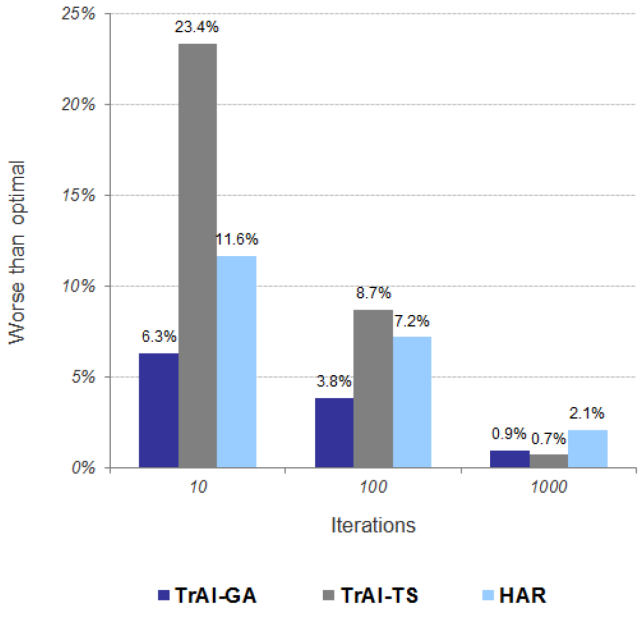

Fig. 7. Average difference between obtained results and optimal result.

As it can be observed, increasing number of iterations causes generating better solutions by the algorithms. However, some of them are influenced by number of iterations more than others. TrAl-GA generates results close to optimal, only 6% worse, even for 10 iterations. Higher number of iterations results in better solutions, but the difference in quality of solutions is not significant. TrAl-TS algorithm presents opposite outcomes. Results obtained after 10 iterations are very weak. Average solution is 23% worse than optimal. It is worth to notice that the solutions are the worst in comparison to other two metaheuristic algorithms. However, for 100 iterations, solutions generated by TrAl-TS are significantly better and are comparable with results obtained by HAR algorithm. After increasing number of iterations to 1000, results generated by TrAl-TS algorithm are better than these obtained by TrAl-GA and HAR. Influence of number of iterations is moderate for HAR algorithm.

For each problem and metaheuristic algorithm, standard deviations from 10, 100, and 1000 iterations were calculated. Data presented in Tab. II confirms observations. The TrAl-TS algorithm is strongly impacted by number of iterations. The lowest influence can be observed for TrAl-GA. It can be also noticed that with increasing the problem size, difference between results obtained with 10, 100, and 1000 iterations is increasing as well.

TABLE II
Standard Deviation of Results

Problem ID	TrAl-GA	TrAl-TS	HAR
1	0.00	2.65	0.00
2	0.00	2.89	1.15
3	0.58	3.61	0.58
4	0.58	4.58	1.53
5	1.53	5.29	2.08
6	3.06	6.11	4.51
7	3.51	8.08	4.51
8	6.81	11.37	6.51
9	0.00	1.73	0.58
10	0.00	2.31	0.58
11	0.00	2.00	1.00
12	0.58	1.73	0.58
13	1.15	4.51	2.52
14	1.53	5.03	2.65
15	2.52	7.02	4.51
16	2.65	10.54	4.51

B. Effectiveness with Respect to Problem Size

In this experiment, different sizes, expressed in number of tasks, of the problem is considered. Sizes of the problem vary from 30 to 100. Each DAG is scheduled on 5 and 10 processors. For each metaheuristic algorithm and number of iterations, average difference between optimal and obtained results is calculated. The average is calculated from results for 5 and 10 processors separately. Table III contains calculated average differences.

The larger is the problem size, the higher difference between optimal solutions and obtained solutions can be observed. However, closer observation allows concluding that problem size affects results of different algorithms in different degree.

For TrAl-GA and HAR algorithms, influence of problem size on results is very high. When the number of tasks exceeds

TABLE III
AVERAGE DIFFERENCE FROM OPTIMAL RESULT FOR DIFFERENT PROBLEM SIZES

	Problem size			
Algorithm	30	40	50	60
TrAl-GA$_{10}$	0%	0%	1%	3%
TrAl-GA$_{100}$	0%	0%	0%	1%
TrAl-GA$_{1000}$	0%	0%	0%	0%
TrAl-TS$_{10}$	21%	17%	17%	16%
TrAl-TS$_{100}$	2%	0%	7%	5%
TrAl-TS$_{1000}$	0%	0%	0%	1%
HAR$_{10}$	0%	5%	5%	5%
HAR$_{100}$	4%	3%	3%	2%
HAR$_{1000}$	0%	0%	0%	0%

	70	80	90	100
TrAl-GA$_{10}$	8%	10%	12%	16%
TrAl-GA$_{100}$	7%	6%	7%	10%
TrAl-GA$_{1000}$	1%	2%	3%	
TrAl-TS$_{10}$	23%	25%	29%	38%
TrAl-TS$_{100}$	10%	14%	17%	15%
TrAl-TS$_{1000}$	0%	1%	0%	2%
HAR$_{10}$	15%	17%	22%	24%
HAR$_{100}$	9%	9%	13%	15%
HAR$_{1000}$	3%	4%	4%	6%

60, results are getting worse very fast. Impact of the problem size on results for TrAl-TS scheme shows different tendency. Although for TrAl-TS results obtained after 10 iterations are much worse than results for TrAl-GA and HAR; the average difference does not increase significantly with increasing size of the problem. For 100 iterations influence of problem size is similar for all three algorithms. However, for 1000 iterations we can observe that quality of results generated by TrAl-GA and HAR is decreasing together with increasing problem size, whereas the quality of solutions obtained by TrAl-TS remains almost at the same level. This can suggest that while increasing number of iterations to 1000, TrAl-TS algorithm saturates for considered size of problems. In other words, further increasing number of iterations does not result in better solutions. For both TrAl-GA and HAR, saturation effect does not occur or it is not so high for number of tasks greater than 70.

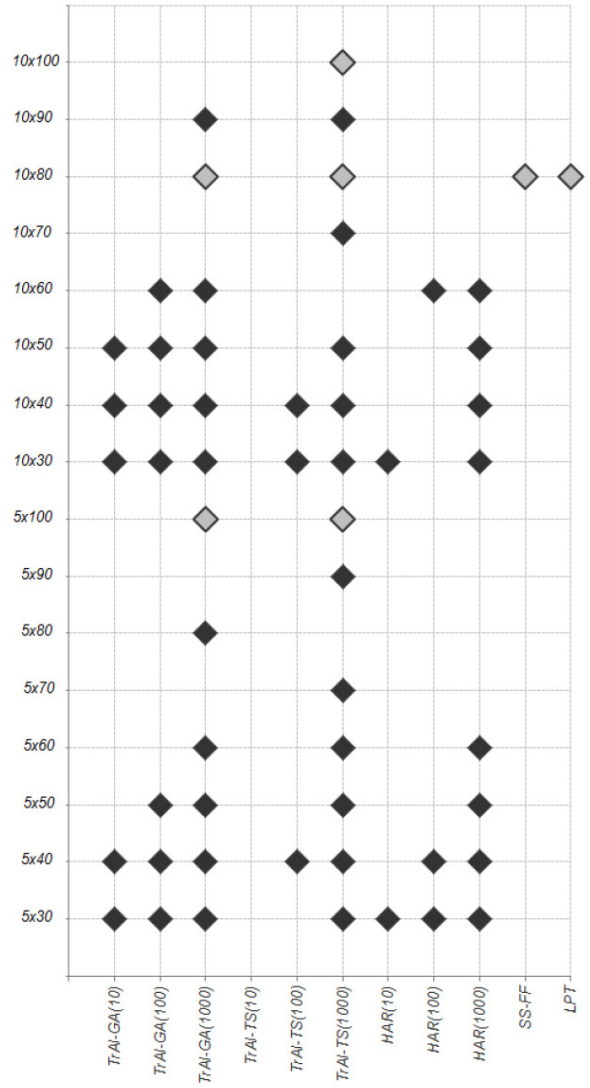

Fig. 8. Map of solutions.

C. Comparison of Effectiveness of Algorithms

Results of the experiment are gathered in the matrix, where columns represent algorithms and rows represent problem instances. This matrix is a solution map (Fig. 8). Mark on intersection of column i and row j shows, that for problem j algorithm i found an optimal solution; or solution better or equal than other algorithms.

As it can be observed, only for problems 8, 14, and 16 optimal solutions were not found. However, generated solutions were close to the optimal. Figure 8 shows weak effectiveness of greedy algorithms. SS-FF and LPT were not able to find optimal solution for any of the considered problems. Results of these algorithms are comparable with the others only for problem number 14. As it was mentioned before, it was one of three problems where none of the algorithms found optimal solution.

Similarly, disappointing results were obtained by TrAl-TS that operates for 10 iterations. The main reason is that TrAl-TS algorithm examines only small space of solutions. In addition, these solutions are close to each other. In order to find better solutions, TrAl-TS has to perform more iterations.

The best results generated in short time were obtained by TrAl-GA. After 10 iterations, TrAl-GA was able to find optimal solutions for 5 out of 16 problems. Any other metaheuristic algorithm did not show such well effectiveness in relatively short time.

For 100 iterations, the best solutions were again generated by TrAl-GA. For 7 out of 16 problems, TrAl-GA found optimal solutions. Effectiveness of TrAl-TS and HAR algorithms for 100 iterations were comparable. It is worth to notice, that for 10 and 100 iterations optimal solution was found only for problem with number of tasks smaller than 60.

Increasing number of iterations to 1000 allows finding optimal solution for problem with size greater than 60. Both TrAl-GA and HAR algorithms were able to generate better results. However, the most significant growth of effectiveness can be observed for TrAl-TS. TrAl-TS was able to find optimal solutions for 11 out of 16 problems; and for next 3 problems,

generated solution was the best. Only TrAl-TS algorithm allowed obtaining optimal solutions for the largest problems. Other two metaheuristic algorithms have incomparable lower effectiveness for 1000 iterations. It is especially notable for HAR algorithm, which did not found any optimal solutions for problems with number of tasks higher than 60.

Observations are summarized in Fig. 9. This figure presents histogram showing specified number of problems, where each algorithm found optimal or the best solution.

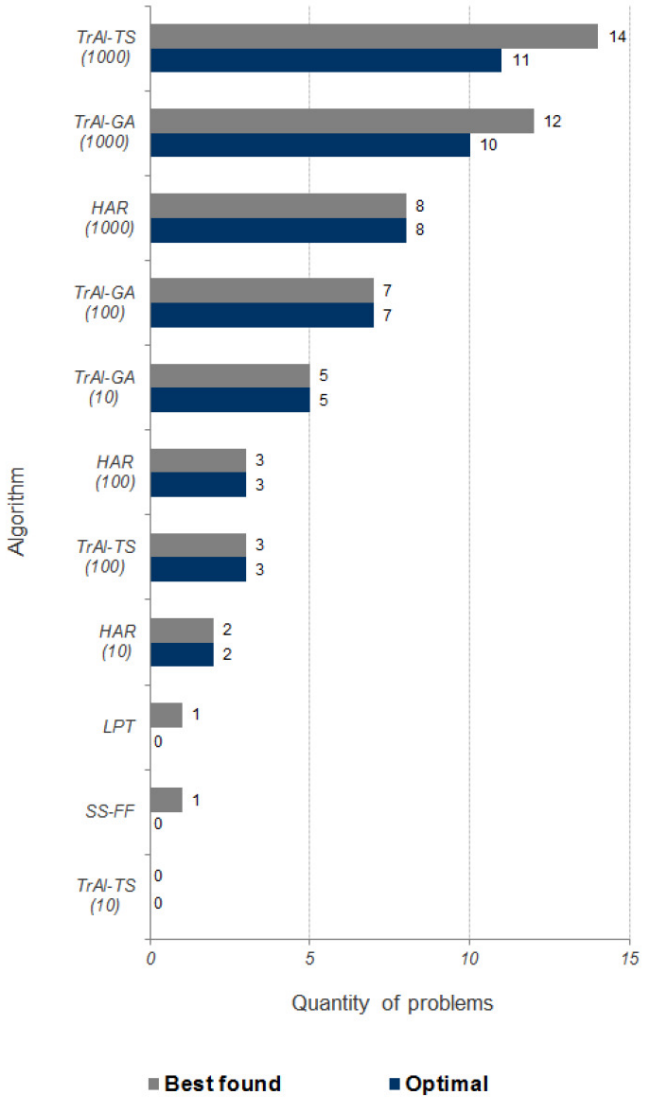

Fig. 9. Histogram of optimal and the best solutions found by each algorithm.

VI. Final Remarks

In this paper, problem of DAG scheduling on parallel executors was considered. New method of solution coding was proposed. Also, two approaches solving considered scheduling problem and using introduced method were proposed. First of them is TrAl-GA and second is TrAl-TS. In order to evaluate effectiveness of new algorithms, experimental investigations were carried out and results were compared to optimal results, solutions generated by well-known HAR algorithm, and to two greedy algorithms SS-FF and LPT.

Observations confirm significant effectiveness of algorithms that use new method of solution coding. The best solutions were obtained by TrAl-GA and TrAl-TS. Both greedy algorithms and HAR algorithm that perform 100 or less iterations were far less effective in comparison two proposed algorithms.

It can be also observed that even for small number of iterations, and therefore in short time, TrAl-GA was able to obtain optimal solutions for over 33% of problems. To compare, HAR algorithm needs 1000 iterations to reach similar effectiveness like TrAl-GA performing for 100 iterations. Therefore, when time is crucial, the best choice is TrAl-GA. On the other hand, when the best solutions are needed; then TrAl-TS algorithm with large number of iterations is a very good choice. It can be observed that TrAl-TS performing 1000 iterations was the best algorithm in ranking and the only one that was able to find optimal solution for the largest problems.

Proposed method of coding solutions is very promising. Experimental results have proved that this approach is very flexible and efficient. It cannot be forgotten that in order to obtain the best efficiency, proper adjusted parameters of the algorithms for considered problem are needed. The great example is TrAl-TS. For this algorithm, higher number of iterations is required to generate good results.

Further work may consider using these algorithms for some well-known benchmarks. It would allow better efficiency evaluation of the algorithms. Good idea would be also testing the proposed solution coding and evaluation methodology using other algorithms and various optimization problems.

References

[1] E. G. Coffman and J. L. Bruno, *Teoria szeregowania zadan (in English Theory of Tasks Scheduling)*. Warsaw: Wydawnictwa Naukowo-Techniczne, 1980.

[2] G. C. I. Centre. (2012, Jun) International grid activities. [Online]. Available: www.gridcomputing.com

[3] D. Zydek, H. Selvaraj, G. Borowik, and T. Luba, "Energy characteristic of processor allocator and network-on-chip," *International Journal of Applied Mathematics and Computer Science*, vol. 21, no. 2, pp. 385–399, 2011, DOI: 10.2478/v10006-011-0029-7.

[4] D. Zydek, H. Selvaraj, L. Koszalka, and I. Pozniak-Koszalka, "Evaluation scheme for noc-based cmp with integrated processor management system," *International Journal of Electronics and Telecommunications*, vol. 56, no. 2, pp. 157–168, 2010, DOI: 10.2478/v10177-010-0021-4.

[5] D. Zydek, N. Shlayan, E. Regentova, and H. Selvaraj, "Review of packet switching technologies for future noc," in *Proceedings of 19th International Conference on Systems Engineering (ICSEng 2008)*, 2008, pp. 306–311, DOI: 10.1109/ICSEng.2008.47.

[6] C. Wang, Z. Li, and S. Zhu, "Minimizing total tardiness on parallel machines based on genetic algorithm," in *Control and Decision Conference, 2008. CCDC 2008. Chinese*, 2008, pp. 165–169, DOI: 10.1109/CCDC.2008.4597291.

[7] H. Jingui and L. Rongheng, "Approximation algorithms on multiprocessor task scheduling," in *Computer Engineering and Technology, 2009. ICCET '09. International Conference*, 2009, pp. 303–307, DOI: 10.1109/ICCET.2009.68.

[8] D. Zydek, G. Chmaj, A. Shawky, and H. Selvaraj, "Location of processor allocator and job scheduler and its impact on cmp performance," *International Journal of Electronics and Telecommunications*, vol. 58, no. 1, pp. 9–14, 2012, DOI: 10.2478/v10177-012-0001-y.

[9] D. Zydek, H. Selvaraj, and L. Gewali, "Synthesis of processor allocator for torus-based chip multiprocessors," in *Proceedings of 7th International Conference on Information Technology: New Generations (ITNG 2010)*, 2010, pp. 13–18, DOI: 10.1109/ITNG.2010.145.

[10] J. C. Liou and M. A. Palis, "A comparison of general approaches to multiprocessor scheduling," in *IPPS '97 Proceedings of the 11th International Symposium on Parallel Processing*, 1997, pp. 152–156, DOI: 10.1109/IPPS.1997.580873.

[11] R. Nossal, "An evolutionary approach to multiprocessor scheduling of dependent tasks," in *the 1st International Workshop on Biologically Inspired Solutions to Parallel Processing Problems*, 1998, pp. 279–287.

[12] M. S. Jelodar, S. N. Fakhraie, F. Montazeri, S. M. Fakhraie, and M. N.Ahmadabadi, "A representation for genetic-algorithm-based multiprocessor task scheduling," in *IEEE Congress on Evolutionary Computation, CEC 2006*, 2006, pp. 340–347, DOI: 10.1109/CEC.2006.1688328.

[13] K. P. Belkhale and P. Banerjee, "Approximate algorithms for the partitionable independent task scheduling problem," in *Proceedings of the 19th International Conference on Parallel Processing*, 1990, pp. 72–75.

[14] X. Cai, C.-Y. Lee, and T.-L. Wong, "Multiprocessor task scheduling to minimize the maximum tardiness and the total completion time," *IEEE Transactions on Robotics and Automation*, vol. 16, no. 6, pp. 824–830, 2000, DOI: 10.1109/70.897792.

[15] G. Łabiak and G. Borowik, "Statechart-based controllers synthesis in FPGA structures with embedded array blocks," *International Journal of Electronics and Telecommunications*, vol. 56, pp. 13–24, 2010, DOI: 10.2478/v10177-010-0002-7.

[16] G. Borowik, M. Rawski, G. Łabiak, A. Bukowiec, and H. Selvaraj, "Efficient logic controller design," in *Fifth International Conference on Broadband and Biomedical Communications (IB2Com)*, Malaga, Spain, Dec. 2010, pp. 1–6, DOI: 10.1109/IB2COM.2010.5723633.

[17] J. Barbosa, C. Morais, R. Nobrega, and A. Monteiro, "Static scheduling of dependent parallel tasks on heterogeneous clusters," in *2005 IEEE International Conference on Cluster Computing*, 2005, pp. 1–8, DOI: 10.1109/CLUSTR.2005.347024.

[18] H. R. Boveiri, "Aco-mts: A new approach for multiprocessor task scheduling based on ant colony optimization," in *Intelligent and Advanced Systems (ICIAS), 2010 International Conference*, 2010, pp. 1–5, DOI: 10.1109/ICIAS.2010.5716203.

[19] R. C. Correa, A. Ferreira, and P. Rebreyend, "Scheduling multiprocessor tasks with genetic algorithms," *IEEE Transactions on Parallel and Distributed Systems*, vol. 10, no. 8, pp. 825–837, 1999, DOI: 10.1109/71.790600.

[20] T. Davidovic and T. G. Crainic, "Benchmark-problem instances for static scheduling of task graphs with communication delays on homogeneous multiprocessor systems," *Computers and Operations Research*, vol. 33, no. 8, pp. 2155–2177, 2006, DOI: 10.1016/j.cor.2005.01.005.

[21] X. Kong, W. Xu, and J. Liu, "A permutation-based differential evolution algorithm incorporating simulated annealing for multiprocessor scheduling with communication delays," in *ICCS'06 Proceedings of the 6th international conference on Computational Science*, vol. Part I, 2006, pp. 514–521, DOI: 10.1007/11758501_70.

[22] A. M. Rahmani and M. Rezvani, "A novel genetic algorithm for static task scheduling in distributed systems," *International Journal of Computer Theory and Engineering*, vol. 1, no. 1, pp. 1–6, 2009, DOI: 10.7763/IJCTE.2009.V1.1.

[23] Y.-K. Kwok and I. Ahmad, "Benchmarking the task graph scheduling algorithms," in *Parallel Processing Symposium, 1998. IPPS/SPDP 1998. Proceedings of the First Merged International ... and Symposium on Parallel and Distributed Processing 1998*, 1998, pp. 531–537, DOI: 10.1109/IPPS.1998.669967.

[24] I. Ahmad and Y.-K. Kwok, "On parallelizing the multiprocessor scheduling problem," *IEEE Transactions on Parallel and Distributed Systems*, vol. 10, no. 4, pp. 414–431, 1999, DOI: 10.1109/71.762819.

[25] M. Drozdowski, "Scheduling multiprocessor tasks - an overview," *European Journal of Operational Research*, vol. 94, no. 2, pp. 215–230, 1996, DOI: 10.1016/0377-2217(96)00123-3.

[26] F. Ghedjati and M.-C. Portmann, "Dynamic heuristics for the generalized job-shop scheduling problem," in *IEEE International Conference on Systems, Man and Cybernetics, 2009, SMC 2009*, 2009, pp. 2562–2567, DOI: 10.1109/ICSMC.2009.5346326.

[27] R. R. A. Rudek, "A flowshop scheduling problem with machine deterioration and maintenance activities," in *Proceedings of the 17th International Conference on Methods and Models in Automation and Robotics*, 2012, pp. 40–45.

[28] A. Y. Zomaya, C. Ward, and B. Macey, "Genetic scheduling for parallel processor systems: comparative studies and performance issues," *IEEE Transactions on Parallel and Distributed Systems*, vol. 10, no. 8, pp. 795–812, 1999, DOI: 10.1109/71.790598.

[29] Z. Michalewicz and D. B. Fogel, *How to Solve It: Modern Heuristics Second Edition*. Springer, 2004.