Zeszyty Naukowe Warszawskiej Wyższej Szkoły Informatyki Nr 9, Rok 7, 2013, s. 15-26

An Ant Colony Optimization Algorithm for Scheduling Parallel Machines with Sequence-Dependent Setup Costs

Ewa Figielska*

Abstract

The paper addresses the problem of scheduling preemptive jobs on parallel unrelated machines in the presence of renewable resource constraints and sequence-dependent setup costs. The objective is to minimize the weighted sum of makespan and setups. The problem is known to be NP-hard. To solve this problem, a heuristic is proposed which uses column generation technique and an ant colony optimization algorithm. The results of a computational experiment indicate that the heuristic is able to produce good results in reasonable computation time.

Keywords: *ant colony optimization, unrelated machines, sequence-dependent setup costs, resource constraints*

1 Introduction

The paper deals with the problem of resource-constrained preemptive scheduling with sequence dependent-setup costs.

We consider a production cell, in which a number of jobs are processed on parallel unrelated machines. We assume that each job is a production lot composed of a large number of identical items, so that it can be treated as a continuously divisible one (jobs are preemptable). Any particular job can be divided into sublots which can be processed separately on the same or different machines. Machines are unrelated which means that the processing times of a job may be different for different machines. Each job to be processed on a machine requires some resources, for example workers or tools. The resources are available in limited quantities at

* Warsaw School of Computer Science.

every moment. When a machine switches from processing one job to another a setup cost is incurred, which depends on the order in which jobs are processed. The objective is the minimization of the cost function consisting of the cost of the makespan (the schedule length) and the total cost of setups. The problem is known to be NP-hard.

The considered problem arises in real-life systems that are encountered in a variety of industries, e.g., in chemical, paper, textile and ceramic industries [1, 2, 3, 4]. The preemptive production arises for example in the textile industry [3], where the processing of any job (the article to be woven) on one of the parallel machines (the looms) may be interrupted (preempted) and resumed on the other machine. Allowing job preemptions results in schedules which are usually much shorter than schedules in which preemptions are forbidden. The example of sequence dependent setups can be found in the ceramic industry, where all products (ceramic tiles) need setups in the molds, glazing lines, kilns and classification lines prior to production [4].

Scheduling problems with setups have received considerable attention from researches during last decades (see [5] for a survey). A lot of papers consider problems with parallel machines and sequence dependent setups (e.g. [6, 7]) but the majority of them do not allow preemptions of jobs. Preemptive scheduling problems has been thoroughly investigated for the cases of parallel machines and renewable resource constraints (e.g. [8, 9]) but without setup consideration. The problem of preemptive scheduling where setups and resource constraints are taken into account and preemptions of jobs are allowed has been considered in [10]. In that paper, a heuristic using a column generation technique and a genetic algorithm was proposed for the case when setup costs do not depend on the sequence of jobs.

In the present paper, we propose a heuristic for solving the resource constrained preemptive scheduling problem with sequence-dependent setup costs. The heuristic proceeds in two steps. First, using a column generation (CG) algorithm, it finds an optimal (with minimal makespan cost) schedule for the problem of scheduling preemptive jobs on unrelated machines with renewable resource constrains but without setup costs. This schedule consists of a number of partial schedules. A partial schedule determines an assignment of jobs to machines for parallel processing during some period of time so that resource constraints are satisfied. Then, the partial schedules are sequenced by means of an ant colony optimization (ACO) algorithm so as to minimize the total cost of setups.

The remainder of the paper is organized as follows. In Section 2, the problem is formally described. Section 3 contains an illustrative example. The heuristic is presented in Section 4. In Section 5 the results of a computational experiment are presented and analysed. Section 6 summarizes the paper.

2 Problem description

The considered problem can be described as follows. There are n preemptive jobs to be processed on m parallel unrelated machines. Each machine can process at most one job at a time and each job can be processed on no more than one machine at a time. Processing a job on a machine may be interrupted at any moment and resumed later on the same or another machine. The processing time of job j ($j = 1, \ldots, n$) is equal to p_{ij} if it is executed on machine i ($i = 1, \ldots, m$). Jobs for their processing, besides the machines, require additional renewable resources. All required resources are granted to a job before its processing begins or resumes and they are returned by the job after finishing its processing or in the case of its preemption. There are l types of renewable resources. A resource of type r ($r = 1, \ldots, l$) is available in an amount limited to W_r units at a time. The total usage of resource r at any moment by jobs simultaneously executed on parallel machines cannot exceed the availability of this resource. Job j during its processing on machine i uses α_{ijr} units of the resource of type r at every moment. When a machine switches from processing job j to processing job k, an appropriate setup of the machine is needed. The cost of this setup is equal to s_{jk}. When job j is the first job processed on a machine, the cost of the setup is equal to s_j^0. The cost of a time unit is equal to c_0. The objective is to minimize the cost function consisting of the makespan cost and the total setup cost.

3 Illustrative example

To illustrate the problem and the solution method we present the following example. Consider the case of the production cell with 2 parallel unrelated machines, 10 jobs and one renewable resource whose availability at any moment, W, is equal to 10. The cost of a time unit equals 100. Job processing times, resource requirements and setup costs are shown in Figure 1.

Figure 2 presents two schedules for this instance. Both the schedules are composed of the same 10 partial schedules, but the orders in which these partial schedules are executed are different for each of the schedule. The schedule shown in Figure 2a has a random order of the partial schedules, while Figure 2b presents the schedule with the order of the partial schedules obtained by the ACO algorithm which aims at the reduction of the total setup cost.

In each of the partial schedules, at most 2 jobs are processed simultaneously and the resource usage does not exceed the resource availability, e.g. in the partial schedule of index 1, jobs 10 and 5 are processed simultaneously and use at every moment 1 and 8 units of the resource, respectively. The resource availability W=10, so the resource constraints are satisfied. The costs associated with the lengths of the schedules in Figures 2a and 2b are the same (equal to 2937).

The schedules shown in Figures 2a and 2b have different orders of the partial schedules, and consequently different total setup costs. The total setup cost for a schedule with a random

order of the partial schedules (Figure 3b) equals 89, while the total setup cost for the schedule with an order of the partial schedules provided by the ACO algorithm (Figure 3b) equals 42. The achieved reduction in the total setup cost is significant (it equals about 53%).

Processing times

job	machine 1	machine 2
1	10	5
2	5	5
3	8	6
4	5	8
5	10	9
6	10	9
7	9	7
8	4	6
9	6	8
10	2	10

Resource requirements

job	machine 1	machine 2
1	5	7
2	2	5
3	6	1
4	2	7
5	8	8
6	3	4
7	3	4
8	4	3
9	1	8
10	1	5

Setup costs between jobs

job	1	2	3	4	5	6	7	8	9	10
1	0	10	7	3	7	3	3	7	9	10
2	2	0	3	3	8	10	1	4	8	1
3	6	1	0	4	3	7	5	8	4	5
4	1	4	4	0	7	2	8	6	5	2
5	6	10	5	2	0	3	1	6	4	9
6	4	1	9	8	8	0	6	10	9	2
7	5	3	6	6	2	9	0	3	6	4
8	3	10	2	1	9	10	9	0	8	8
9	1	8	1	7	3	4	1	8	0	5
10	2	8	2	3	5	1	6	4	3	0

Setup costs if a job starts as first on a machine

job	
1	10
2	10
3	10
4	9
5	5
6	1
7	8
8	1
9	10
10	1

Figure 1. Data for an illustrative example

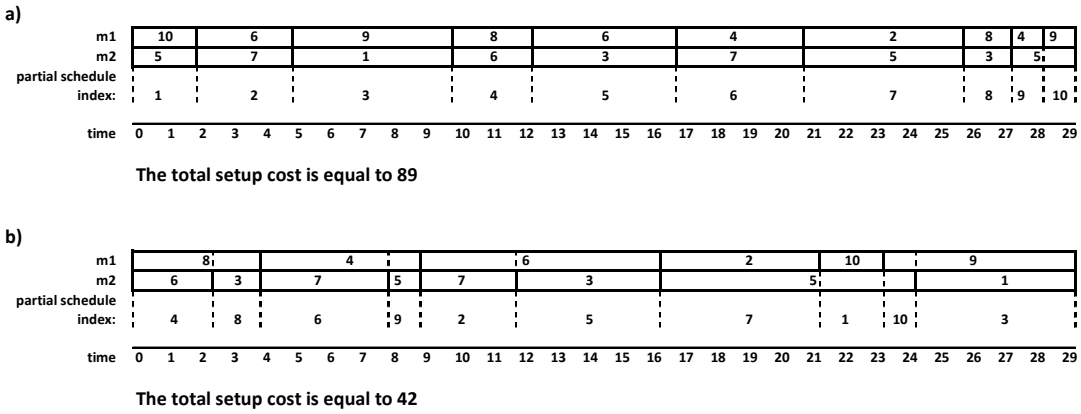

Figure 2. The schedules: a) the schedule before the reduction in the total setup cost, b) the schedule with the total setup cost reduced by the ACO algorithm.

In Figure 2b, we can see that the partial schedules are placed so that setup costs between consecutive jobs are small. Also, partial schedules containing the same job are often processed one by one without placing another partial schedule between them. For example, consider job of index 4 executed in partial schedules 6 and 9. In the schedule in Figure 2a, this job is preceded by jobs 6 and 8 (executed, respectively, in partial schedules 5 and 8). So, the total

setup cost incurred by switching from job 6 to job 4 and from job 8 to job 4 is equal to 8+1=9. In the schedule in Figure 2b, job 4 is preceded only by job 8 (as partial schedules 6 and 9 are executed one after another without interruption) which results in setup cost equal to 1.

4 The heuristic

The heuristic proceeds in two steps which can be outlined as follows.

1. A set of partial schedules satisfying resource constraints is found by means of the column generation algorithm so as to minimize the makespan cost.
2. The ant colony optimization algorithm finds the order of the partial schedules so as to minimize the total setup cost.

4.1 Column generation algorithm

The theoretical basis of the CG algorithm has been provided by Dantzig and Wolfe [11] (for applications of the CG technique see e.g. [12, 13, 14]).

Using column generation technique, we avoid the difficulty of explicitly generating all columns of the problem, i.e. all possible partial schedules (in our work a column corresponds to a partial schedule) by working with only a subset of the columns and adding new columns as needed (when they improve the solution).

The CG algorithm is an iterative procedure which starts from an initial set of columns and, in subsequent iterations, solves a linear programming (LP) problem with all columns generated so far, and then generates a new column.

Let us denote by P_β the partial schedule of index $\beta, \beta \in B$, where B is the set of the indices of all possible partial schedules. Partial schedule P_β is determined by its duration Δ_β and the values of v_{ij}^β $(i = 1, \dots, m,\ j = 1, \dots, n)$ representing the assignment of jobs to machines, where $v_{ij}^\beta = 1$ if job j is processed on machine i in partial schedule P_β, and $v_{ij}^\beta = 0$, otherwise.

At each iteration of the CG algorithm, the cost of the makespan (makespan is equal to the sum of the partial schedule durations) is minimized by solving the following LP problem:

$$\min c_0 \sum_{\beta \in \tilde{B}} \Delta_\beta \tag{1}$$

subject to:

$$\sum_{\beta \in \tilde{B}} \Delta_\beta \sum_{i=1}^{m} \frac{v_{ij}^\beta}{p_{ij}} = 1,\ j = 1, \dots, n \tag{2}$$

$$\Delta_\beta \geq 0,\ \beta \in \tilde{B} \tag{3}$$

where Δ_β ($\beta \in \tilde{B}$) are decision variables. $\tilde{B}$ denotes a subset of set B, which contains the indices of the already generated columns. The values of v_{ij}^{β} ($\beta \in \tilde{B}, i = 1, \dots, m, j = 1, \dots, n$) are fixed. They are given in advance before the first iteration and then calculated in each of the successive iterations. Constraints (2) ensure that all jobs are completed.

In a successive iteration, the current set $\tilde{B}$ may be extended by a new index β' ($\beta' \in B \backslash \tilde{B}$). In order to find the assignment of jobs to machines in the new column with index β' the following problem is solved:

$$\max \sum_{i=1}^{m} \sum_{j=1}^{n} \frac{\pi_j^* v_{ij}^{\beta'}}{p_{ij}} \tag{4}$$

subject to:

$$\sum_{j=1}^{n} v_{ij}^{\beta'} \le 1,\ i = 1,\dots,m, \tag{5}$$

$$\sum_{i=1}^{m} v_{ij}^{\beta'} \le 1,\ j = 1,\dots,n, \tag{6}$$

$$\sum_{j=1}^{n} \sum_{i=1}^{m} \alpha_{ijr} v_{ij}^{\beta'} \le W_r,\ r = 1,\dots,l, \tag{7}$$

$$v_{ij}^{\beta'} \in \{0,1\},\ i = 1,\dots,m,\ j = 1,\dots,n, \tag{8}$$

where π_j^* ($j = 1,\dots,n$) are the optimal values of dual variables corresponding to constraints (2).

Constraints (5) and (6) ensure that in each partial schedule, respectively, each machine works on at most one job at a time and each job is processed on no more than one machine at a time. Due to constraint (7), in each partial schedule, the usage of each resource at every moment does not exceed its availability. A new iteration of the CG algorithm begins if $\sum_{i=1}^{m} \sum_{j=1}^{n} \pi_j^* v_{ij}^{\beta'} / p_{ij} - 1 > 0$, otherwise the optimal solution is found and the algorithm stops.

4.2 Ant colony optimization algorithm

Ant colony optimization algorithms have been proposed by Dorigo at all. [15] in 1996, and since then they have been successfully applied to a wide range of engineering problems (see e.g. [16]).

ACO algorithms are inspired by social behavior of ants, which are able to find the shortest path from the nest to the source of food. Ants communicate with each other by means of pheromone trails. When ants leave their nest in search for a food source, they move randomly in different directions depositing pheromone wherever they go (see Figure 3). The ants that have found the food, carry some of it back to the nest and start again following their trail and

depositing more pheromone. So, on the shortest path to the food more pheromone will be deposited than on the other paths. Other ants, which leave the nest later, most likely choose the path with the greatest concentration of the pheromone. Moreover, pheromone evaporates over time, disappearing from not used paths. So, in the end, all the ants are moving along the shortest path from the nest to the food source.

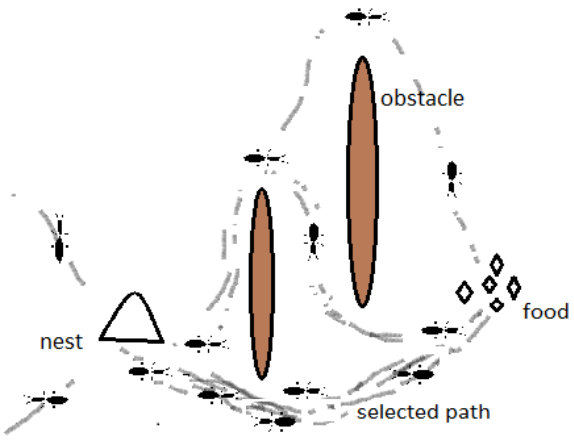

Figure 3. The shortest path to the food

The structure of the ACO algorithm used in this paper can be described as follows:

1. Set the initial pheromone values.
2. Generate and evaluate an initial population of ants.
3. While the termination condition is not met, do the following:
   a. Update the pheromone values.
   b. Update the solution represented by each ant on the basis of the pheromone concentration.
   c. Evaluate each ant.
4. Return the best solution found.

In this paper, the ACO algorithm is used for finding a sequence of the partial schedules (created by the CG algorithm) which minimizes the total setup cost. A candidate solution is and ant which represents a sequence of the indices of the partial schedules. An initial population of ants is randomly generated. The total setup cost is taken as an objective function. Each ant is evaluated according to an objective function. In consecutive iterations of the ACO algorithm, the pheromone matrix $[\tau_{ij}]$ is determined in the following way. The pheromone value τ_{ij} is calculated according to the following formula:

$$\tau_{ij} = \rho\tau'_{ij} + \Delta\tau_{ij}, \tag{9}$$

where i ($i = 1,...,b$) is the index of the position in the sequence represented by an ant, b is the number of partial schedules, j ($j = 1,...,b$) is the index of the partial schedule, ρ is pheromone evaporation rate, τ'_{ij} is the pheromone value in the previous iteration and $\Delta\tau_{ij}$ is a change in the pheromone value.

The value of $\Delta\tau_{ij}$ is calculated as follows:

$$\Delta\tau_{ij} = \frac{1}{K_{ij}} \sum_{k=1}^{psize} \begin{cases} \frac{Q}{obj(k)} & \text{if value } j \text{ is chosen by ant } k \text{ to be put in position } i \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

where $psize$ is the size of the population of ants, $obj(k)$ is the objective function value (the total setup cost) for ant k, K_{ij} is the number of ants in the population for which value j is chosen to be put in position i. Q is a given constant.

Once the pheromone is updated, the ants (the sequences of the partial schedule indices) are changed in the following way. In each ant, the partial schedule index j is chosen to be put in position i with the probability which is given by the following expression:

$$P_{ij} = \frac{\tau_{ij}}{\sum_{j=1}^{b} \tau_{ij}} .$$

Consequently, in each iteration, the best ants are built.

In our implementation, the values of the parameters of the ACO algorithm are as follows: $psize = 5, \rho = 0.25, Q = 1000$. The algorithm stops after 10000 iterations.

The example of the pheromone matrix and one ant is shown in Figure 5. The pheromone matrix contains pheromone values calculated according to equations (9) and (10). Given these values, a new ant is constructed, so that a partial schedule for which the pheromone value is greater is more likely chosen to be put in a given position in the ant. For example, while considering position 1 in the ant, partial schedules of indices 4 and 5 (for which $\tau_{ij} = 15.23$ and 13.02, respectively) have a greater chance to be placed in this position than the other partial schedules (for which $\tau_{ij} = 2.17$). The figure shows the ant constructed on the basis of the given pheromone matrix.

The order of the partial schedules represented by this ant is: 4, 3, 5, 2, 1.

The pheromone matrix:

Partial schedule indices	Indices of the positions in the sequence				
	1	2	3	4	5
1	2.17	14.79	5.31	2.17	13.53
2	2.17	3.10	2.17	14.52	2.17
3	2.17	16.10	2.17	12.82	2.17
4	15.23	2.17	20.39	15.33	2.17
5	13.02	2.17	17.72	12.82	14.67

The ant:

4	3	5	2	1

Figure 4. The pheromone matrix and the ant

5 Computational experiment

In this section, we present the results of a computational experiment which was carried out to evaluate the proposed heuristic. Tests were performed for problems with the number of jobs n = 25 and 50, the number of machines m = 2, 4 and 6, and one resource. Processing times, resource requirements and setup costs were generated from a discrete uniform distribution over the interval [1,10]. The resource availability W was set at 10. All presented results are average values over 5 problems.

The results of a computational experiment are shown in Figures 5 and 6. Figure 5 presents the reduction in the total setup cost achieved by the ACO algorithm defined as follows:

$$\theta = \frac{S - S_{ACO}}{S} \times 100\%,$$

where S is the total setup cost for the best solution (ant) in the initial population, and S_{ACO} is the total setup cost for the best solution found by the ACO algorithm.

In Figure 5, we can see that the reduction in the total setup cost, θ, achieved by the ACO algorithm is quite significant. The greatest value of θ (47.19%) was obtained for problems with 10 jobs and 2 machines. θ decreases with increasing the number of jobs. For problems with 50 jobs and 2 machines the average value of θ equals 22.42%. The values of θ are less

the number of machines than by the number of jobs, though also in this case, we observe that θ decreases as the number of machines grows.

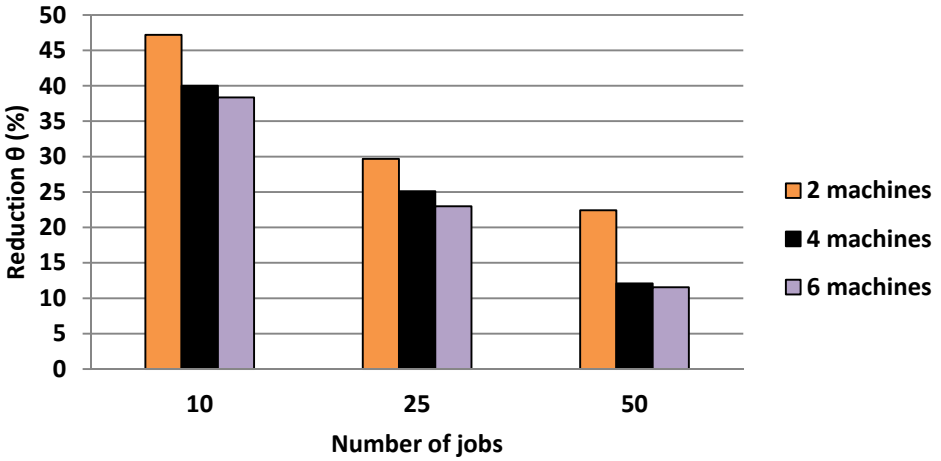

Figure 5. The reduction in the total setup cost achieved by the ACO algorithm

The computation times of the heuristic are small – they do not exceed 6 seconds for all the considered problems. The computation time increases with the number of jobs. The number of machines affects the computation times very slightly.

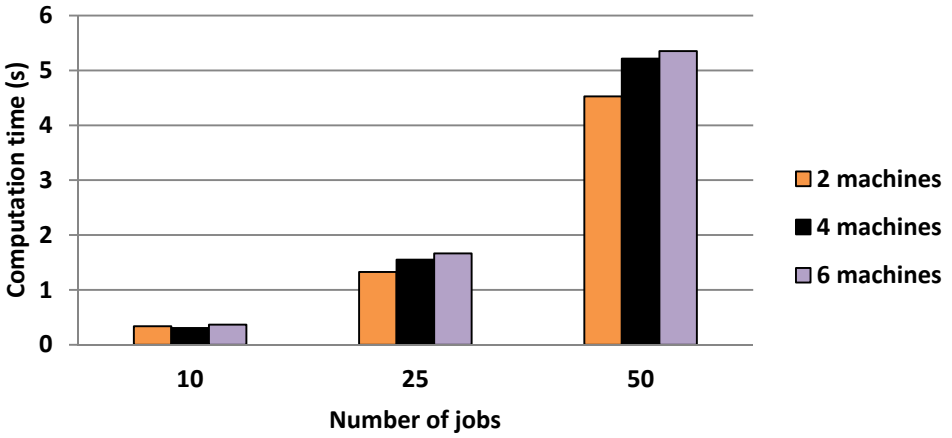

Figure 6. The computation times of the heuristic

6 Summary and concluding remarks

In the paper, the heuristic has been presented which joins the column generation technique and the ant colony optimization algorithm to solve the problem of resource constrained preemptive scheduling on unrelated machines with sequence-dependent setup costs. The aim was to minimize the sum of makespan and setup costs. The results of the computational experiment have shown that the ant colony optimization algorithm significantly reduces the total setup costs in the schedule with minimal cost makespan produced by the column generation algorithm.

References

[1] Fortemps O., Ost C., Pirlot M., Teghem J., Tuyttens D., *Using metaheuristics for solving a production scheduling problem in a chemical firm*, "International Journal of Production Economics" 1996, Vol. 46-47 (13), s. 13-26

[2] Sherali H.D., Sarin S.C., Kodialam M.S, *Models and algorithm for a two stage production process*, "Production Planning and Control" 1990, Vol.1 (1), s. 27-39

[3] Serafini P., *Scheduling jobs on several machines with the job splitting property*, "Operations Research" 1996, Vol. 44 (4), s. 617-628

[4] Ruiz R., Maroto C., *A genetic algorithm for hybrid flowshops with sequence dependent setup times and machine eligibility*, "European Journal of Operational Research" 2006, Vol. 169, s. 781-800

[5] Allahverdi A., Ng C.T., Cheng T.C.E., Kovalyov M.Y., *A survey of scheduling problems with setup times or costs*, "European Journal of Operational Research" 2008, Vol. 187, s. 985-1032

[6] Rocha P.L., Ravetti M.G., Mateus G.R., Pardalos P.M., *Exact algorithms for scheduling problem with unrelated parallel machines and sequence and machine-dependent setup times*, "Computers & Operations Research" 2008, Vol. 35 (4), s. 1250-1264

[7] Driessel R., Mönch L., *Variable neighborhood search approaches for scheduling jobs on parallel machines with sequence-dependent setup times, precedence constraints, and ready times*, "Computers & Industrial Engineering" 2011, Vol. 61 (2), s. 336-345

[8] Słowiński R., *Two approaches to problems of resource allocation among project activities – A comparative study*, "Journal of Operational Research Society" 1980, Vol. 31, s. 711-723

[9] Werra D. De, *On the two-phase method for preemptive scheduling*, "European Journal of Operational Research" 1988, Vol. 37, s. 227-235

[10] Figielska E., *Preemptive scheduling with changeovers: using column generation technique and genetic algorithm*, "Computers and Industrial Engineering" 1999, Vol. 37, s. 3-66

[11] Dantzig G.B., Wolfe P., *Decomposition principle for linear programs*, "Operations Research" 1960, Vol. 8, s. 101-111

[12] Gilmore P.C., Gomory R.E., *A linear programming approach to the cutting-stock problem*, "Operations Research" 1961, Vol. 9, s. 849-859
[13] Chen Z.-L., Lee C.-Y., *Parallel machine scheduling with a common due window*, "European Journal of Operational Research" 2002, Vol. 136, s. 512-527
[14] Figielska E., *A genetic algorithm and a simulated annealing algorithm combined with column generation technique for solving the problem of scheduling in the hybrid flowshop with additional resources*, "Computers and Industrial Engineering" 2009, Vol. 56, s. 142-151
[15] Dorigo M., Maniezzo V., Colorni A., *Ant system: optimization by a colony of cooperating agents*, "IEEE Transactions on Systems, Man and Cybernetics" 1996, Vol. 26 (1), s. 29-41
[16] Mohan B.C., Baskaran R., *A survey: Ant Colony Optimization based recent research and implementation on several engineering domain*, "Expert Systems with Applications" 2012, Vol. 39, s. 4618-4627

Zastosowanie algorytmu mrówkowego do szeregowania zadań na maszynach równoległych z uwzględnieniem kosztów przezbrojeń zależnych od kolejności zadań

Streszczenie

Artykuł dotyczy zagadnienia szeregowania zadań podzielnych na równoległych dowolnych maszynach z uwzględnieniem ograniczeń na dostępność zasobów odnawialnych oraz kosztów przezbrojeń zależnych od kolejności wykonywania zadań. Celem jest minimalizacja ważonej sumy czasu trwania harmonogramu i przezbrojeń. Zagadnienie należy do klasy problemów NP-trudnych. W celu jego rozwiązania, zaproponowany został algorytm heurystyczny, wykorzystujący technikę generacji kolumn, oraz algorytm mrówkowy. Wyniki eksperymentu obliczeniowego wskazują, że algorytm ten jest zdolny dostarczyć dobrej jakości wyniki w rozsądnym czasie.

***Słowa kluczowe**: algorytm mrówkowy, dowolne maszyny, koszty przezbrojeń zależne od kolejności zadań, ograniczenia zasobowe*