Scheduling Independent Tasks: Bee Colony Optimization Approach

Tatjana Davidović, Milica Šelmić, Dušan Teodorović

***Abstract*— The problem of static scheduling of independent tasks on homogeneous multiprocessor systems is studied in this paper. The problem is solved by the Bee Colony Optimization (BCO). The BCO algorithm belongs to the class of stochastic swarm optimization methods. The proposed algorithm is inspired by the foraging habits of bees in the nature. The BCO algorithm was able to obtain the optimal value of objective function in all small to medium size test problems. The CPU times required to find the best solutions by the BCO are acceptable.**

I. Introduction

THE resourceful use of multiprocessor systems highly depends on scheduling tasks to be performed on processors. The level of service in multiprocessor systems and the system's total costs primarily depend on a chosen schedule [7]. In this paper, we study the static problem of scheduling independent tasks on homogeneous multiprocessor systems. The word "static" means that we know in advance the total number of tasks, as well as the duration of each task. We assume that multiprocessor system contains *m* identical processors. The scheduling of *n* tasks to processors consists in assigning tasks to processors, as well as determining their starting times. Even this simple variant of the scheduling problem is known to be NP-hard [9]. These problems are usually solved by various heuristic algorithms or procedures based on meta-heuristic rules.

The problem of scheduling independent tasks to multiprocessor systems is solved in this paper by the Bee Colony Optimization [12-14]. The preliminary experimental results show that the proposed algorithm can generate high quality solutions for randomly generated test examples.

This paper is organized in the following way. The considered scheduling problem is described in Section II. Section III contains the description of the Bee Colony Optimization meta-heuristics. The Bee Colony Optimization for the scheduling problem is given in Section IV. Section V contains test results. Conclusion is given in Section VI.

Tatjana Davidović is with the Mathematical Institute of SASA, Belgrade, Serbia; tanjad@mi.sanu.ac.rs.

Milica Šelmić and Dušan Teodorović are with the Faculty of Transport and Traffic Engineering, Belgrade, Serbia; m.selmic@sf.bg.ac.rs, dusan@sf.bg.ac.rs.

II. The Problem of Static Scheduling of Independent Tasks on Homogeneous Multiprocessors

In this paper we consider the following problem. Let $T = \{1,2,...,n\}$ be a given set of independent tasks, and $P = \{1,2,...m\}$ represents the set of identical processors. We denote by l_i the *a priori* known processing time of task i (i = 1,2,…,n). All tasks are mutually independent and each task can be scheduled to any processor. All given tasks must be executed. Task should be scheduled to exactly one processor and processors can execute one task at a time. Task preemption is not allowed. The goal is to find schedule of tasks to processors in such a way as to minimize the completion time of all tasks (the so called *makespan*).

The considered scheduling problem could be graphically represented by Gantt diagram (Fig. 1).

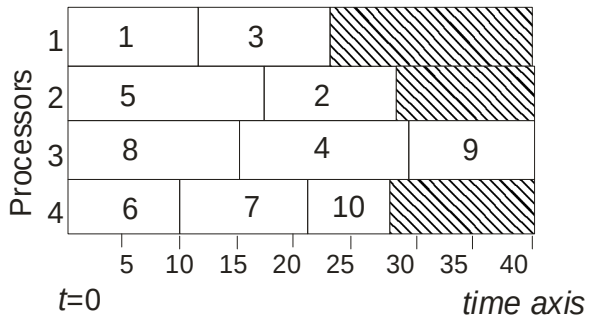

Fig. 1. Gantt diagram: Scheduled tasks to processors.

On the horizontal axis in the diagram given in Fig. 1 we measure the time. The white rectangles in the Gantt diagram represent tasks. Shaded rectangles are used to denote idle time on a processor. The starting time of a task is determined by the completion times of all tasks already scheduled to the same processor. The total completion time (makespan) for the schedule shown in the Fig. 1 equals 40 time units (the completion time of task 9 scheduled to processor 3). Any schedule that has completion time less than 40 time units is considered better. The goal is to find a schedule of tasks to processors that has shortest makespan. In order to present mathematical programming formulation of the problem, let us introduce the binary variables defined in the following way:

$$x_{ij} = \begin{cases} 1, & \text{if task } i \text{ is scheduled to processor } j, \\ 0, & \text{otherwise.} \end{cases} \quad (1)$$

The considered scheduling problem can be formulated as a linear program in the following way [15]:

Minimize

$$y \tag{2}$$

subject to

$$\sum_{j=1}^{m} x_{ij} = 1, \qquad 1 \le i \le n \tag{3}$$

$$y - \sum_{i=1}^{n} l_i x_{ij} \ge 0 \qquad 1 \le j \le m \tag{4}$$

$$x_{ij} \in \{0,1\}, \qquad 1 \le i \le n, 1 \le j \le m \tag{5}$$

The objective function that should be minimized represents the total completion time of all tasks – makespan *y*. Each task *i* should be scheduled to one and only one potential processor *j* (constraint (3)). The makespan *y* is computed as the maximum over all processor's computation times, and processing time of a processor is defined as a sum of processing times of all tasks scheduled to that processor. This is described by the constraints (4). Constraints (5) show binary nature of the variables x_{ij}.

There are several exact algorithms in the literature with the goal to solve small to medium size problems to optimality [8,15,16]. A lot of approximation algorithms (heuristic and meta-heuristic ones) for solving this problem have been proposed (for example [6,10,11,17]). Constructive heuristic implementing "largest processing time first" rule (LPT) proposed in [10] falls into the so called "list scheduling" algorithms group. It consists of two steps: 1) tasks are sorted in decreasing order of their processing times and then 2) each task is assigned to the least loaded processor (ties are broken by the minimal index of task and/or processor). MultiFit algorithm [6] uses the fact that this scheduling problem can be formulated as the bin packing problem and produces better results than the LPT algorithm but with increasing in the computational complexity. The Tabu Search approach has been proposed in [17] and has also been used in [16] to improve the performance of an exact algorithm.

III. Bee Colony Optimization

The Bee Colony Optimization (BCO) is a meta-heuristic for solving combinatorial optimization problems. The BCO algorithm belongs to the class of stochastic swarm optimization methods. The proposed algorithm is inspired by the foraging habits of bees in the nature. The communication systems between individual insects contribute to the configuration of the ''collective intelligence" of the social insect colonies. The term ''Swarm intelligence", that denotes this ''collective intelligence" has come into use [1], [2], [3], [4]. Swarm intelligence [4] is the part of Artificial intelligence based on studying actions of individuals in various decentralized systems.

The BCO is inspired by bees' behavior in the nature. The basic idea behind the BCO is to create the multi agent system (colony of artificial bees) capable to successfully solve difficult combinatorial optimization problems. The artificial bee colony behaves partially alike, and partially differently from bee colonies in nature. We will first describe the behavior of bees in nature, then give a short summary of a general Bee Colony Optimization algorithm and afterwards, in the next section, we introduce and describe in details the algorithm developed for the considered scheduling problem.

A. Bees in the Nature

In spite of the existence of a large number of different social insect species, and variation in their behavioral patterns, it is possible to describe individual insects' as capable of performing a variety of complex tasks [5]. The best example is the collection and processing of nectar, the practice of which is highly organized. Each bee decides to reach the nectar source by following a nestmate who has already discovered a patch of flowers. Each hive has a so-called dance floor area on which the bees that have discovered nectar sources dance, in that way trying to convince their nestmates to follow them. If a bee decides to leave the hive to get nectar, it follows one of the bee dancers to one of the nectar areas. Upon arrival, the foraging bee takes a load of nectar and returns to the hive relinquishing the nectar to a food store. After it relinquishes the food, the bee can (a) abandon the food source and become again uncommitted follower, (b) continue to forage at the food source without recruiting the nestmates, or (c) dance and thus recruit the nestmates before the return to the food source. The bee opts for one of the above alternatives with a certain probability. Within the dance area, the bee dancers "advertise" different food sources.

B. BCO Algorithm

Lučić and Teodorović [12-14] were first who used basic principles of collective bee intelligence in solving combinatorial optimization problems. The BCO is a population based algorithm. Population of the *artificial bees* searches for the optimal solution. Every artificial bee generates one solution to the problem. The algorithm consists of two alternating phases: *forward pass* and *backward pass*. During each forward pass, every bee is exploring the search space. It applies a predefined number of moves, which construct and/or improve the solution, yielding to a new solution.

Having obtained new partial solutions, the bees return to the hive and start the second phase, the so-called backward pass. During the backward pass, all bees share information about their solutions. In nature, bees would perform a dancing ritual, which would inform other bees about the amount of food they have found, and the proximity of the patch to the hive. In the search algorithm, the bees announce the quality of the solution, i.e. the value of objective function. During the backward pass, every bee decides with a certain probability whether it will advertise its solution or not. The bees with better solutions have more chances to

advertise their solutions. The remaining bees have to decide whether to continue to explore their own solution in the next forward pass, or to start exploring the neighborhood of one of the solutions being advertised. Similarly, this decision is taken with a probability, and therefore better solutions have higher probability of being chosen for exploration.

The two phases of the search algorithm, forward and backward pass, are performed *iteratively*, until a stopping condition is met. The possible stopping conditions could be, for example, the maximum total number of forward/backward passes, the maximum total number of forward/backward passes without the improvement of the objective function, etc.

The BCO algorithm parameters whose values need to be set prior the algorithm execution are as follows:

B - The number of bees in the hive;

NC - The number of constructive moves during one forward pass.

At the beginning of search process, all bees are in the hive. The following is the pseudo code of the BCO algorithm:

1. Initialization: every bee is set to an empty solution;
2. For every bee do the forward pass:
    a) Set $k = 1$; //counter for constructive moves //in the forward pass;
    b) Evaluate all possible constructive moves;
    c) According to evaluation, choose one move using the roulette wheel;
    d) $k = k + 1$; If $k \leq NC$ Go To step b;
3. All bees are back to the hive; // backward pass starts;
4. Sort the bees by their objective function value;
5. Every bee decides randomly whether to continue its own exploration and become a recruiter, or to become a follower (bees with higher objective function value have greater chance to continue its own exploration);
6. For every follower, choose a new solution from recruiters by the roulette wheel;
7. If the stopping condition is not met Go To step 2;
8. Output the best result.

IV. The BCO Approach to Scheduling problem

In this paper, we propose the BCO heuristic algorithm tailored for problem of scheduling tasks on homogeneous processors. As of the authors' knowledge this is the first implementation of Swarm intelligence to a given problem.

At the beginning of a scheduling process, we assume that all bees are in the hive. The hive is an artificial location, it is not connected either to processors or to the tasks. We allow every artificial bee to fly out from the hive and to generate *NC* constructive moves. After that, every bee returns to the hive. Bees exchange information about the quality of the partial solutions generated. After obtaining full information about all partial solutions generated by all bees, every bee decides whether to abandon its partial solution and become again uncommitted follower, or dance and thus recruit the nestmates before flying again from the hive.

A. Constructive moves in forward pass

Each constructive move in the forward pass consists of choosing a task-processor pair. Following the idea of LPT algorithm [10], that it is better to choose longer tasks first and then use the shorter ones to refine the schedule, we set up p_i (the probability that specific bee chooses task *i*) to:

$$p_i = \frac{l_i}{\sum_{k=1}^{K} l_k}, \quad i=1,2,\ldots,n \tag{6}$$

where:

l_i - processing time of the *i*-th task;

K - the number of "free" tasks (not previously chosen).

Obviously, tasks with a higher processing time have a higher chance to be chosen. Using relation (6) and a random number generator, we determine a task to be chosen by each bee.

Ones the task to be scheduled is selected corresponding processor should be chosen. Since our goal is to minimize maximum (over all processors) running time it is obvious that processors with a lower value of the current running times should have a higher chances to be chosen. Let us denote by p_j the probability that specific bee chooses processor *j*. We assume that the probability of choosing processor *j* equals:

$$p_j = \frac{V_j}{\sum_{k=1}^{m} V_k}, \quad j = 1,2,\ldots,m \tag{7}$$

where:

$$V_j = \frac{\max F - F_j}{\max F - \min F}, \quad j = 1,2,\ldots,m \tag{8}$$

and:

F_j - running time of processor *j* based on tasks already scheduled to it;

max *F*- maximum over all processors running times (based on already scheduled tasks);

min *F* - minimum over all processors running times (based on already scheduled tasks).

Therefore, *Vj* represents normalized value for the running time of corresponding processor. Using relation (7) and a

random number generator, we select a processor for previously chosen task.

Within a single forward pass, each bee has to determine *NC* task-processor pairs. In total, *B* bees choose *B*NC* task-processor pairs after each forward pass. When scheduling tasks to processors is done for all pairs within a single forward pass, we update processors' running times and start the backward pass.

B. Bee's partial solutions comparison mechanism

All bees return to the hive after generating the partial solutions. All these solutions are then evaluated by all bees. (The latest time point of finishing the last task at any processor characterizes every generated partial solution).

Let us denote by C_b (b=1, 2,..., B) the latest time point of finishing the last task at any processor in the partial solution generated by the b-th bee. We denote by O_b normalized value of the time point C_b, i.e.:

$$O_b = \frac{C_{\max} - C_b}{C_{\max} - C_{\min}}, \quad b = 1,2,...,B \tag{9}$$

where C_{min} and C_{max} are respectively the smallest and the largest time point among all time points produced by all bees. The probability that b-th bee (at the beginning of the new forward pass) is loyal to the previously discovered partial solution is calculated in this paper in the following way:

$$p_b^{u+1} = e^{-\frac{O_{\max} - O_b}{u}}, \quad b = 1,2,...,B \tag{10}$$

where u is the ordinary number of the forward pass.

Using relation (10) and a random number generator, every artificial bee decides to become uncommitted follower, or to continue flight along already known path (Fig. 2).

Let us discuss relation (10) that we propose in a more details. The better the generated partial solution (higher O_b value), the higher the probability that the bee will be loyal to the previously discovered partial solution. The greater the ordinary number of the forward pass, the higher the influence of the already discovered partial solution. This is expressed by the term u in the nominator of the exponent (relation (10)). In other words, at the beginning of the search process bees are "more brave" to search the solution space. The more forward passes they make, the bees have less courage to explore the solution space. The more we are approaching the end of the search process, the more focused the bees are on the already known solutions.

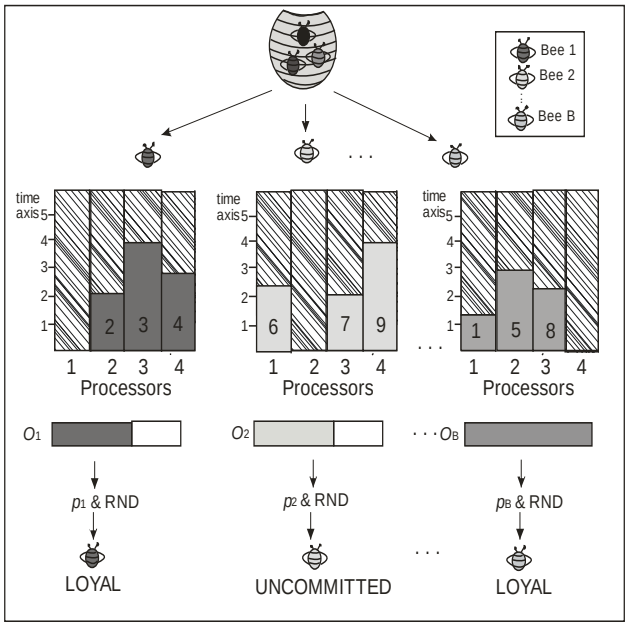

Fig. 2. Comparison of partial solutions after third forward pass, *NC*=1.

C. Recruiting Process

In the case when at the beginning of a new stage bee does not want to expand previously generated partial solution, the bee will go to the dancing area and will follow another bee (Fig. 3).

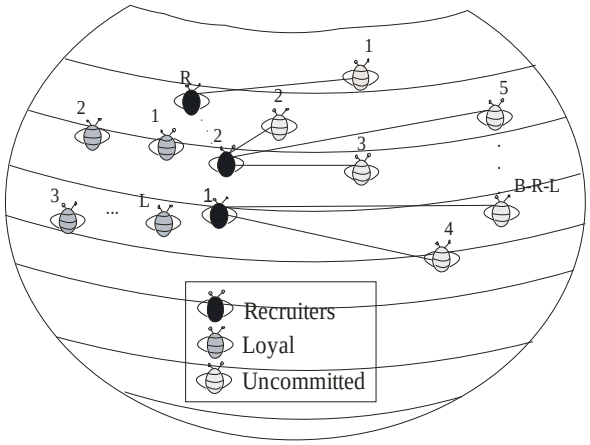

Fig. 3. Recruiting of uncommitted followers.

Within the dance area the bee-dancers (recruiters) "advertise" different partial solutions. We have assumed in this paper that the probability the recruiter b's partial solution will be chosen by any uncommitted bee equals:

$$p_b = \frac{O_b}{\sum_{k=1}^{R} O_k}, \quad b=1,2,\ldots,R \tag{11}$$

where:

O_k - objective function value of the k-th advertised solution;

R - the number of recruiters.

Using relation (11) and a random number generator, every uncommitted follower join one bee dancer (recruiter). Recruiters fly together with a recruted nestmates in the next forward pass along the path discovered by the recruiter. At the end of this path all bees are free to independently search the solution space and generate the next iteration constructive moves.

V. Test Problems

The proposed algorithm was tested on a various test problems. We denote respectively by *NT* and *NP* the number of tasks and the number of processors. The problem parameters range from instances with *NT* = 10 up to the instances with *NT*= 100. In all cases we set *NP* = 4. The BCO algorithm parameters are set to the following values: The total number of bees *B* engaged in the search process was equal to 10; *NC* - the number of moves (generated task-processor pairs) during one forward pass was equal to 1; the number of iterations *I* within one run was equal to 100.

Test problems are generated using the random task graph generators proposed in [7]. We compared the obtained BCO results with the optimal solution. The comparison results are shown in the Tables I, II, III, IV and V. Within all tables, BCO represents objective function value obtained by the BCO algorithm; OPT denotes the optimal makespan obtained by using ILOG AMPL and CPLEX 11.2 optimization software; CPU time shows the time required by BCO algorithm to obtain the optimal solution; I stands for the number of iteration until optimal solution was reached.

TABLE I
THE COMPARISON OF THE BCO RESULTS WITH OBJECTIVE FUNCTION OPTIMAL VALUES FOR SMALL SIZE PROBLEMS (*NT*=10 AND *NT*=20, *NP*=4)

Test problem	BCO	OPT	BCO time (sec)	I
It10_1	34	34	0.1629	2
It10_2	36	36	0.1757	4
It10_3	33	33	0.1482	1
It10_4	30	30	0.1692	3
It10_5	27	27	0.1687	2
It20_1	69	69	0.2439	7
It20_2	70	70	0.2101	4
It20_3	73	73	0.2415	6
It20_4	70	70	0.2983	10
It20_5	63	63	0.2135	3

TABLE II
THE COMPARISON OF THE BCO RESULTS WITH OBJECTIVE FUNCTION OPTIMAL VALUES FOR SMALL SIZE PROBLEMS (*NT*=30 AND *NP*=4)

Test problem	BCO	OPT	BCO time (sec)	I
It30_1	128	128	0.5332	10
It30_2	152	152	0.4736	10
It30_3	154	154	0.4266	5
It30_4	159	159	0.4018	9
It30_5	132	132	0.1709	1
It30_6	148	148	0.6379	10
It30_7	157	157	0.3158	6
It30_8	168	168	0.1811	1
It30_9	141	141	0.3339	7
It30_10	180	180	2.8587	64
It30_11	155	155	0.2821	3
It30_12	151	151	0.7191	18
It30_13	170	170	0.2173	2
It30_14	140	140	0.2479	3

TABLE III
THE COMPARISON OF THE BCO RESULTS WITH OBJECTIVE FUNCTION OPTIMAL VALUES FOR SMALL SIZE PROBLEMS (*NT*=40 AND *NP*=4)

Test problem	BCO	OPT	BCO time (sec)	I
It40_1	211	211	0.6525	9
It40_2	226	226	0.2156	2
It40_3	205	205	0.1972	1
It40_4	206	206	0.3729	5
It40_5	182	182	0.2189	2
It40_6	198	198	0.3799	3
It40_7	169	169	0.2733	2
It40_8	201	201	0.3163	3
It40_9	235	235	0.3325	4
It40_10	210	210	0.6283	8
It40_11	229	229	0.3020	5
It40_12	204	204	0.5392	7
It40_13	201	201	0.5023	8
It40_14	153	153	0.6017	5

TABLE IV
THE COMPARISON OF THE BCO RESULTS WITH OBJECTIVE FUNCTION OPTIMAL VALUES FOR MEDIUM SIZE PROBLEMS (*NT*=50 AND *NP*=4)

Test problem	BCO	OPT	BCO time (sec)	I
It50 70	212	212	1.0776	5
It50 80	196	196	0.5637	1
It50 80_1	234	234	1.1848	4
It50 80_2	337	337	1.7368	8
It50 80_3	216	216	0.8077	3
It50 80_4	276	276	0.7472	2
It50 80_5	128	128	0.8814	4
It50 80_6	167	167	1.8514	8

TABLE V
THE COMPARISON OF THE BCO RESULTS WITH OBJECTIVE FUNCTION OPTIMAL VALUES FOR MEDIUM SIZE PROBLEMS (*NT*=1000 AND *NP*=4)

Test problem	BCO	OPT	BCO time (sec)	I
It100_40_1	493	493	3.8002	11
It100_40_2	782	782	4.9475	8
It100_40_3	478	478	2.7898	6
It100_40_4	483	483	1.7391	5
It100_40_5	271	271	0.6710	1
It100_40_6	340	340	1.1896	2
It100_50_1	471	471	1.5869	3
It100_60_1	465	465	1.9314	3

The proposed BCO algorithm was able to obtain the optimal value of objective function in all test problems. The CPU times required to find the best solutions by the BCO are acceptable. In other words, the BCO was able to produce optimal solutions within very small number of iteration. We expect that our implementation will produce high quality solutions for large size problems, too. All the tests were performed on AMD Sempron (tm) Processor with 1.60 GHz and 512 MB of RAM under Windows OS.

VI. Conclusion

The Bee Colony Optimization (BCO) has been successfully applied to the problem of static scheduling of independent tasks on homogeneous multiprocessor systems. The preliminary results show that BCO was able to obtain optimal solution for all tested randomly generated small to

medium size examples. We expect that our implementation will produce high quality solutions for large size problems, too. The proposed approach probably represents good addition to the existing meta-heuristic approaches dealing with the scheduling problem.

There are no theoretical results at the moment that could support proposed approach. Usually, development of various meta-heuristic was based on experimental work in initial stage. Good experimental results than motivate researchers to try to produce some theoretical background. The concept proposed in this paper is not exception in this sense.

REFERENCES

[1] G. Beni, "The concept of cellular robotic system," in *Proc. of the 1988 IEEE International Symposium on Intelligent Control*, IEEE Computer Society Press, Los Alamitos, CA , pp. 57–62.

[2] G. Beni, and J. Wang, "Swarm intelligence," in *Proc. of the Seventh Annual Meeting of the Robotics Society of Japan*, RSJ Press, Tokyo, 1989, pp. 425–428.

[3] G. Beni, and S. Hackwood, "Stationary waves in cyclic swarms," in: *Proc. of the 1992 International Symposium on Intelligent Control*, IEEE Computer Society Press, Los Alamitos, CA, pp. 234–242.

[4] E. Bonabeau, M. Dorigo, and G. Theraulaz, *Swarm intelligence*. Oxford University Press, Oxford, 1997.

[5] S. Camazine, and J. Sneyd, "A model of collective nectar source by honey bees: self-organization through simple rules," *Journal of Theoretical Biology*, vol. 149, pp. 547- 571, 1991.

[6] E. G. Coffman Jr, M. R. Garey, and D.S. Jonson, "An application of bin-packing to multiprocessor scheduling," *SIAM Journal on Computing*, vol. 7, pp. 1-17, 1978.

[7] T. Davidović, and T. G. Crainic, "Benchmark-problem instances for static scheduling of task graphs with communication delays on homogeneous multiprocessor systems," *Computers & Operations Research*, pp. 2155–2177, 2006.

[8] M. Dell'Amico, and S. Martello, "Optimal scheduling of tasks on identical parallel processors", *ORSA Journal on Computing*, vol. 7, pp. 191-200, 1995.

[9] M. R. Garey, and D. S. Johnson, *Computers and intractability: a guide to the theory of NP-completeness*. W. H. Freeman and Company, 1979.

[10] R.L. Graham, "Bounds on multiprocessor timing anomalies," *SIAM Journal on Applied Mathematics*, vol. 17, pp. 416-429, 2006.

[11] M. Haouari, A. Gharbi, and M. Jemmali, "Tight bounds for the identical parallel machine scheduling problem," *International Transaction in Operational Research*, vol. 13, pp. 529-548, 2006.

[12] P. Lučić, and D. Teodorović, "Bee system: modeling combinatorial optimization transportation engineering problems by swarm intelligence," in *Preprints of the TRISTAN IV Triennial Symposium on Transportation Analysis*. Sao Miguel, Azores Islands, 2001, pp. 441-445.

[13] P. Lučić, and D. Teodorović, "Vehicle routing problem with uncertain demand at nodes: the bee system and fuzzy logic approach", in *Fuzzy Sets based Heuristics for Optimization*, Verdegay JL, Eds. Physica Verlag: Berlin Heidelberg, 2002, pp. 67-82.

[14] P. Lučić, and D. Teodorović, "Computing with bees: attacking complex transportation engineering problems," *International Journal on Artificial Intelligence Tools*, vol. 12, pp. 375-394, 2003.

[15] E. Mokotoff, "An exact algorithm for the identical parallel machine scheduling problem," *European Journal of Operational Research*, vol. 152, pp.758-769, 2004.

[16] S. Shakeri, and R. Logendran, "A mathematical programming-based scheduling framework for multitasking environments," *European Journal of Operational Research*, vol. 176, pp. 193-209, 2007.

[17] A. Thesen, "Design and evaluation of tabu search algorithm for multiprocessor scheduling," *Journal of Heuristics*, vol. 4, pp. 141-160, 1998.