BULLETIN OF THE POLISH ACADEMY OF SCIENCES
TECHNICAL SCIENCES, Vol. 62, No. 4, 2014
DOI: 10.2478/bpasts-2014-0087

Comparison of firefly and cockroach algorithms in selected discrete and combinatorial problems

J. KWIECIEŃ* and B. FILIPOWICZ

AGH University of Science and Technology, 30 Mickiewicza Ave., 30-059 Krakow, Poland

Abstract. In recent years, newer algorithms inspired by nature have been created and used to solve various problems. Therefore, in the paper we present the application of firefly and cockroach algorithms to optimize two queueing systems and permutation flow shop problems with the objective of minimizing the makespan. The article briefly describes these algorithms to solve selected problems and their results. Because these algorithms were originally developed for continuous optimization problems, we introduce a new formula to transform the position of ith individual to solve the discrete problems.

Key words: queueing systems, flow shop scheduling, firefly algorithm, cockroach algorithm.

1. Introduction

Recently, many NP-hard optimization problems have been successfully analysed by the nature-inspired metaheuristics based on existing mechanisms of a biological phenomenon of nature. Animals adapt to current conditions of living as best they can. Some species specialize in hunting, others have developed the ability of quick escape or camouflage. Many population-based algorithms are extensively studied in the literature. Particle swarm optimization algorithm [1], ant colony optimization algorithm [2, 3], bee algorithm [4] and artificial bee colony algorithm [5, 6] have been widely used in solving many problems. Firefly and cockroach algorithms are the latest metaheuristics. These algorithms have not been tested in many optimization problems yet, so we present their usefulness to solve the selected problems. The firefly algorithm (FA) was proposed by Xin-She Yang at Cambridge University in 2008 [7]. It is inspired by the behavior of fireflies. Their flashing light generated by a process of bioluminescence may serve as an element of courtship rituals or warning signals. The cockroach swarm optimization algorithm (CSO) was presented by Chen and Tang in [8].

Manufacturing companies from different sectors of the economy are looking for solutions that allow for optimization of processes in many areas of their activities. Many production systems are the flow types. They are characterized by the fact that the final product is produced by passing through successive stages of processing, where the explicit realization of operations (machining, assembling) takes place. In computer systems, the specific calculations are performed by dedicated processors. Technological progress generates more efficient and more complex production systems and IT systems, which consist of more and more machines, processors. This causes an increase in the number of performed tasks, often more complex and thus requires more and better ways to optimize processes. In scheduling problems, even with a relatively small number of jobs and machines, it is not possible to find optimal solutions in accessible time. The best known way seems to be the use of approximate algorithms, which are a compromise between the quality of the resulting solution and computation time.

The problems of queueing systems emerge in many issues related to the subject of operations research. Queueing theory has aroused intense interest of many mathematicians, economists and engineers. Its principal objective is to develop methods to determine the values of basic indicators that characterize the service process and the selection of the optimal structure and organization of the service. Hence, it is necessary to use various methods, including these inspired by nature, to determine the optimal structure.

The rest of the paper is organized as follows: Sec. 2 presents the idea of the firefly and cockroach algorithms, while Sec. 3 describes the examples of discrete and combinatorial problems, such as the permutation flow shop and optimization of queueing systems. The results of the metaheuristics used to solve these problems, including their analysis and comparison, are described in Sec. 4. Finally, Sec. 5 summarizes the conclusions.

2. Nature-inspired metaheuristics

In solving NP-hard problems, we are on the line by two factors: the optimal solution and the time at which a solution is reached. The achievement of a result (which is fairly good approximation of an ideal) in real time is often more desirable than looking for the best result in a very long time. The nature-inspired metaheuristics meet the goal.

2.1. Firefly algorithm. The firefly algorithm uses the behavior of fireflies, which tend to the light source (optimal solution) and interact with light emitted by them (each firefly moves towards a brighter partner), in proportion to the mutu-

*e-mail: kwiecien@agh.edu.pl

al distance. The brightness of each firefly perceived by others depends on the distance from the light source. Fireflies also turn to the brighter partner, which ensures that the information about the brightness of the light source is propagated not only by the same source, but also by insects.

In optimization problems, values of the objective function correspond to the searched problem space, while achieving the extreme values of the objective function is associated with achievement of the light sources by fireflies. The objective function of a given optimization problem is based on differences in light intensity. Each firefly changes its position iteratively.

Xin-She Yang outlined three principles that determine the behavior of fireflies in the algorithm [7, 9]:

- All fireflies are of the same sex, each firefly can be attracted by the other regardless of their gender.
- The attractiveness of firefly is proportional to its brightness. The lighter firefly, the more attractive to the others and from any two fireflies, less luminous firefly tend towards the brighter one. If the brighter one is not around, firefly moves at random.
- Brightness of a firefly is determined by the objective function. For maximization problems, the brightness can be assumed directly proportional to the value of the objective function.

The attractiveness of firefly β is a factor that determines the power of attraction of fireflies. It depends on the mutual distance between the fireflies. Attractiveness is given by [7]:

$$\beta(r) = \beta_0 e^{-\gamma r^m}, \qquad m \geq 1, \tag{1}$$

where β_0 denotes the maximum attractiveness (at $r = 0$) and γ is the light absorption coefficient, which controls the decrease of the light intensity.

The distance between the fireflies, has the significant influence on the direction of their movement. The distance between two fireflies i and j at positions x_i and x_j can be represented as [7]:

$$r_{ij} = \|x_i - x_j\| = \sqrt{\sum_{k=1}^{D} (x_{i,k} - x_{j,k})^2}, \tag{2}$$

where $x_{i,k}$ is the k-th component of the spatial coordinate x_i of i-th firefly and D denotes the number of dimensions.

The movement of firefly is affected by several factors: location of the firefly (it decides on the relationship between the fireflies), attractiveness and randomness. The movement of a firefly i is determined by the following form [7]:

$$x_i = x_i + \beta_0 e^{-\gamma r_{ij}^2}(x_j - x_i) + \alpha \left(\text{rand} - \frac{1}{2}\right), \tag{3}$$

where the first factor is the current position of a firefly i, the second one denotes a firefly's attractiveness and the last factor is used for the random movement if there is not any brighter firefly (rand is a random number generator uniformly distributed in the range $< 0, 1 >$). Depending on the choice of the parameter we can control the influence of randomness on the total displacement. For most cases $\alpha \in (0, 1)$, $\beta_0 = 1$, $\gamma =< 0.1, 10 >$. Parameter γ is responsible for the rate of FA convergence [7, 9].

The firefly algorithm is executed as follows [7]:

```
STEP 1: Initialize algorithm's parameters and generate population of fireflies ($n$ – number of fireflies, $i = 1, ..., n$, $\beta_0$, $\gamma$, $\alpha$, MaxIt – maximum number of iterations); define the objective function $f(x)$, $x = (x_1, \ldots, x_D)^T$.
STEP2: Calculate light intensities; light intensity of firefly $I_i$ at $x_i$ is determined by the value of objective function $f(x_i)$.
STEP 3: For each iteration ($1, ..., MaxIt$) do:
    For $i = 1 : n$
      For $j = 1:n$
          If ($I_j > I_i$) move firefly $i$ towards firefly $j$ in
          $D$-dimension according to Eq. (3); End if
          Obtain attractiveness, which varies with
          distance $r$ according to Eq. (1)
          Find new solutions and update light intensity
      End for $j$
    End for $i$
Rank the fireflies, firefly with the maximum light intensity is chosen as the potential optimal solution
STEP 4: If a predefined stopping criterion is met then output the results, otherwise go back to STEP 3. Find the firefly with the highest light intensity, visualization.
```

2.2. Cockroach swarm algorithm. Cockroach swarm optimization (CSO) is based on the observation of the behavior of cockroaches looking for food. The way in which these insects acquire food, focus the attention of researchers. The whole phenomenon of "hunting" is an accidental finding food, assisted by the movement of designated routes through experience. The CSO algorithm uses a number of different cockroach's behaviors, such as going in swarms, scattering or escape from the light, which is particularly significant because of the prevention of situations that the algorithm stucks in a local optimum. Cockroaches scatter when the random vector of light appears, which guarantees their continuous movement [8, 10, 11].

In CSO algorithm, each cockroach X_i individually goes to a local optimum P_i in the area of its visibility. The best cockroach in the range of visibility is the one which tends to the nearest local extreme. The best local optimum is a global optimum at the end of the cycle, denoted as P_g [8, 10].

Analyzing "going in swarm", in the new cycle all the strongest cockroaches form small swarms and follow the global optimum. Other cockroaches will go well for the strongest, for which the strongest cockroaches become local optimum P_i. At any time there is a possibility that the cockroach follower in a small swarm, will be strongest if it finds a better solution. This is because the cockroaches do not follow in the same way as their local optima. A lonely cockroach (within its scope of visibility) is a local optimum for itself. Dispersion of cockroaches means that at a time, each individual will be randomly dispersed in order to maintain the current individual diversity. Dispersion is a simulation of the appearance of light. From time to time, we also have to deal with the ruthless behavior, when the current best individual replaces a randomly chosen individual. In fact, in case of shortage of food, the stronger eats the weaker (if it is much "stronger") [8, 10].

The cockroach swarm optimization algorithm can be presented as follows [8, 10]:

STEP 1: Initialize algorithm's parameters and randomly generate population (*step*, the visual distance of cockroach *visual*, D – space dimension, n – number of cockroach individuals, *MaxIt* – maximum number of iterations, w – the inertia factor from range $< 0,\ 1 >$), the i-th individual represents a vector $X_i = (x_{i1}, x_{i2}, \ldots, x_{iD})$, $i = 1, 2, \ldots, n$.

STEP 2: Search P_i (the optimal individual within the visual scope of X_i) and P_g (the global optimal individual).

STEP 3: Implement behavior of "going in swarm" as follows:

- If a cockroach X_i is locally most powerful (local optimum), then the cockroach goes to the global optimum P_g, which is the most powerful cockroach:

$$X_i = w^* X_i + step^* \mathrm{rand} < 0, 1 >^* (P_g - X_i) \ \text{ for } \ X_i = P_i.$$

- Otherwise, the cockroach goes to a local optimum of P_i, which is the locally most powerful cockroach:

$$X_i = w^* X_i + step^* \mathrm{rand} < 0, 1 >^* (P_i - X_i) \ \text{ for } \ X_i \neq P_i.$$

Update P_g.

STEP 4: Implement dispersing behavior by the equation:

$$X_i = w^* X_i + \mathrm{rand}(1, D), \ \text{ update } \ P_g.$$

STEP 5: Implement ruthless behavior by $X_k = P_g$, or $X_k = 0$; k is a random integer within $< 1,\ n >$.

STEP 6: If a predefined stopping criterion is met then output the results, otherwise go back to STEP 2.

3. Examples of optimization problems

In the paper, we consider only two problems: structural optimization of two queueing systems and permutation flow shop problem as examples of discrete and combinatorial optimization.

3.1. Permutation flow shop problem. Flow shop problem is one of the models of jobs scheduling on dedicated machines, in which jobs are performed by all machines in the same order. Single job consists of operations that are executed on separate machines, according to the imposed order [12, 13].

The permutation flow shop problem can be formulated as follows: there is a set of jobs $J = \{1, 2, \ldots, n\}$ and a set of machines $M = \{1, 2, \ldots, m\}$. The job j from the set J consists of a sequence of operations $O_{j,1}, O_{j,2}, ..., O_{j,m}$, the operation $O_{j,k}$ corresponds the executed job j on machine k ($k = 1, 2, \ldots, m$) at a specified time $t_{j,k}$. Jobs from the set J are performed on machines from the set M with the following assumptions [12, 13]:

- at any given time, each machine can perform one job at most,
- each job can be performed in a time of at most one machine,
- operation started can not be interrupted,
- performing jobs on the machine k may be started only after completion of processing on the machine $k-1$,
- additionally, the order of execution of jobs by each of the machines is the same.

Each sequence of jobs can be described by permutation $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$, in which $\pi(i)$ represents the job inserted at the i-th place. Denoting by $C(\pi(i), j)$ the completion times and the maskespan (the length of the scheduling) by $C_{\max}(\pi)$, we can obtain these parameters from Eq. (4). Finally, the objective is to find a permutation from the set of all permutation for which the makespan (the length of the scheduling) is minimal.

$$\begin{gathered} C(\pi(1), 1) = t_{\pi(1),1} \\ C(\pi(i), 1) = C(\pi(i-1), 1) + t_{\pi(i),1}, \qquad i = 2, ..., n \\ C(\pi(1), j) = C(\pi(1), j-1) + t_{\pi(1),j}, \quad j = 2, ..., m \\ C(\pi(i), j) = \max\{C(\pi(i-1), j), C(\pi(i), j-1)\} + t_{\pi(i),j} \\ i = 2, ..., n, \qquad j = 2, ..., m \\ C_{\max}(\pi) = C(\pi(n), m). \end{gathered} \tag{4}$$

3.2. Queueing systems. In considered models we have a Poisson arrival process with arrival rate λ. The service times are exponentially distributed with parameter μ, and traffic intensity ρ is the ratio of arrival λ to service rate μ [14–16].

The M/M/m/FIFO/m+N queueing system with finite capacity and impatient customers. Sometimes it may happen that customers should be served before their deadlines. Hence, we will consider m-server queueing system with limited waiting room (N) with FIFO queueing discipline and limited waiting time in the queue (T_w, it is assumed to be with an exponential distribution with parameter δ). The job is waiting in the queue for a time T_w. If newly arriving customers find $m + N$ customers in systems, they are lost [16].

The steady-state probability of the system being empty is obtained from:

$$\pi_0 = \left[\sum_{k=0}^{m} \frac{\rho^k}{k!} + \frac{\rho^m}{m!} \sum_{r=1}^{m+N} \frac{\rho^r}{\prod\limits_{n=1}^{r} \left(m + n \dfrac{\delta}{\mu} \right)} \right]^{-1}. \tag{5}$$

The probability that jobs will be lost because of exceeding the time limit is given by:

$$\pi_w = \left(\frac{\delta}{\lambda} \cdot \frac{\rho^m}{m!} \cdot \sum_{r=1}^{m+N} \frac{r \rho^r}{\prod\limits_{n=1}^{r} \left(m + n \dfrac{\delta}{\mu} \right)} \right) \cdot \pi_0. \tag{6}$$

The probability that the newly arriving customers are lost because there are $m + N$ jobs in system has the following form:

$$\pi_{m+N} = \frac{\dfrac{\rho^{m+N}}{m!}\pi_0}{\prod\limits_{n=1}^{N}\left(m+n\dfrac{\delta}{\mu}\right)}. \tag{7}$$

The probability that jobs will be lost is as follows:

$$\pi_l = \pi_w \cdot \pi_{m+N}. \tag{8}$$

The M/M/m/FIFO/N/F closed queueing system with finite population of N jobs. In the presented model it is assumed that the number of jobs does not exceed the number N and the number of servers is m. In the industrial processes, it is a typical case that the jobs for service come from a limited number of objects [15]. The steady-state probability of no jobs in systems is given by:

$$\pi_0 = \left[\sum_{i=0}^{m}\frac{N!}{i!(N-i)!}\rho^i + \sum_{j=m+1}^{N}\frac{N!}{m!(N-j)!m^{j-m}}\rho^j\right]^{-1}. \tag{9}$$

We may calculate the mean number of jobs in a system from the following expression:

$$\overline{K} = \pi_0 N! a,$$

$$a = \sum_{i=0}^{m}\frac{i}{i!(N-i)!}\rho^i + \sum_{j=m+1}^{N}\frac{j}{m!(N-j)!m^{j-m}}\rho^j. \tag{10}$$

4. Optimization by nature-inspired metaheuristics

In the literature, we often encounter scheduling problems, but the problems of structure optimization of queueing systems are treated quite superficially. A comparison of many optimization techniques dedicated to solution of permutation flow shop problem with the makespan criterion can be found in [17]. Due to the complexity of flow shop scheduling, we have good solution techniques, for example Johnson's algorithm for two machine problems, constructive algorithm CDS or NEH [13, 18] Many studies describe the tabu search approach in solving the flow shop problem, for example [19–21]. Nature-inspired metaheuristics were also used to solve the flow shop problem. The application of particle swarm optimization was presented in [22], while ant colony algorithms can be found in [23]. The article aims to examine whether firefly and cockroach algorithms can be used to solve such problems. All experiments were performed in Matlab under Windows 7 operating system.

4.1. Optimization of permutation flow shop problem. The firefly algorithm and cockroaches swarm optimization algorithm were originally developed for continuous optimization problems. However, many optimization tasks are discrete. The FA and CSO algorithms can be used to solve discrete optimization problems, but such application requires some additional assumptions. Recall the appropriate scheduling of jobs (permutation) is a solution of the flow shop problem. A suitable encoding scheme of jobs sequence (equivalent to positions of entities) is very important issue in applying considered algorithms to flow shop scheduling problem.

Permutation is often encoded in the form of a vector, which can be represented as a $X_i^l = (x_{i1}^l, x_{i2}^l, \ldots, x_{ij}^l, \ldots, x_{in}^l)$, where i – number of individual, j – the position of the job, l – number generation. Each value x_{ij}^k must be a positive integer. Unfortunately, as a result of FA and CSO algorithms to optimize these types of problems, we receive vector consisting of real numbers, which is difficult to interpret. Therefore, in the paper, we encoded a schedule in other way - in the form of 0-1 matrix of ranked probabilities ($n \times n$ matrix, n is the number of jobs), as shown in [24]. Let $X_i^l = [x_{ijk}^l]$ be the matrix (equivalent to the position of the individual), where each value x_{ijk}^l represents possibility assignment j-th job of i-th individual in k-th position of the schedule at iteration l [24]. This matrix has binary numbers and shows to which position (in schedule) the job is assigned. Therefore, the probability that the j-th job will be ranked at position k takes the value 0 or 1. If this probability is equal to 1 then the j-th job has been ranked at the position k-th, otherwise it is equal to 0. According to [24], we assume that all individuals (fireflies and cockroaches) are represented by binary variables (see Table 1).

Table 1
Definition of entity for sequence (2 1 3 4)

		Position (k)			
		1	2	3	4
Job (j)	1	0	1	0	0
	2	1	0	0	0
	3	0	0	1	0
	4	0	0	0	1

As a result of original firefly and cockroach algorithms, we can get the real numbers in a schedule. In original firefly algorithm, after movement fireflies and in original CSO algorithm, after "going in swarm" and "dispersing" procedures, the obtained values in the matrix would adopt the form of real numbers outside the range <0, 1>. Therefore, these values should be converted to the probabilities values (range $< 0, 1 >$) by using the appropriate formula, e.g. the sigmoid function in [24]. It should be noted, that each entity assigns jobs to position in schedule based on its changes of probabilities. In the next step, we have to change the obtained matrix of probabilities to the binary matrix. Thus, the maximum probability (P_{best}) is searched for each job in the matrix of probabilities. The value of P_{best} is converted to 1 if the position k with the highest probability is selected and the job j is placed in this position (in this position we have not yet assigned any job). If this position is already occupied, the second probability is searched (in order from the largest) and its value is changed to 1. All other values in the row take the value "0". Detailed description of FA application to flow shop problem with conversion procedure of the firefly position (from real numbers to the change of the probabilities) is presented in [24].

We propose to transform each element of the matrix (x_i) by the following formula:

$$X_i = \frac{x_i - x_i^{\min}}{x_i^{\max} - x_i^{\min}}, \tag{11}$$

where X_i achieves the result belonging to $< 0, 1 >$, $x_i^{\max}$ and $x_i^{\min}$ are the defined maximum and minimum value of x_i.

In these algorithms, the space dimension is equal to number of jobs. The initial populations of individuals are generated according to the formula:

$$x_i = x_i^{\min} + \text{rand} \cdot (x_i^{\max} - x_i^{\min}). \tag{12}$$

The used firefly-scheduling mechanism is outlined as follows:

```
Input:
   -  parameters of test instance (number of jobs,
      number of machines, processing times);
   -  parameters of FA ($n$, $\beta_0$, $\gamma$, $\alpha$, MaxIt - maximum
      number of iterations as stop criterion),
Output: global best position (global best schedule =
      minimization of makespan);
Begin
  for $i$ = 1 to $n$ do // $n$ – number of fireflies
     generate randomly (each firefly
     represents one scheduling solution, the solution is
     written as 0-1 matrix of probabilities);
     compute the makespan $C_{max}$
  end
  while (number_iteration < MaxIt) do
     for $i$ = 1 to $n$ do
       for $j$ = 1 to $n$ do
          compare the fitness value ($C_{max}$)
          if $C_{max}(j)<C_{max}(i)$ then // firefly j is brighter
             move firefly $i$ towards $j$
          else move firefly $i$ randomly;
          end if
          //Construct a solution (a sequence of jobs) by
          //following convert rule (CR):
          for all rows in matrix do
             transform the obtained real numbers (position)
             to the values of probabilities using Eq.(11)
          end for
          for all rows in matrix do
             find $P_{best}$
             replace $P_{best}$ by 1
             values of the others elements in row and
             column (where $P_{best}$ was detected) replace by 0
          end for
          calculate the fitness ($C_{max}$)
       end for $j$
     end for $i$
     Determine the best candidate sequence as a current
     sequence (with the best $C_{max}$)
  end while
  Return the best solution as sub-optimal schedule
end
```

Similarly, in the case of CSO algorithm, we construct a new solution with change the cockroach position from continuous to discrete value (see "convert rule" presented above). The pseudo-code of the used cockroach-scheduling mechanism can be stated as follows:

```
Input:
   -  parameters of test instance (number of jobs,
      number of machines, processing times);
   -  parameters of CSO ($n$, step, visual, MaxIt, $w$)
Output: global best position (global best schedule =
      minimization of makespan);
Begin
  for $i$ = 1 to $n$ do // $n$ – number of cockroaches
     generate randomly (each entity
     represents one scheduling solution, the solution is
     written as 0-1 matrix of probabilities);
     compute the makespan $C_{max}$
  end
  find the best solution (with min $C_{max}$) in initial
  population
  while stop condition not met do
   //{going in swarm}
   for $i$ to $n$
      for $j$ to $n$
         if $C_{max}(j)<C_{max}(i)$ // (cockroach j is better than
         cockroach i; within its visual)
            move cockroach $i$ towards $j$;
            construct a solution (a sequence of jobs) by
            convert rule (CR)
            if new solution is better than local optimum
            update local optimum (cockroach $i$ is a
            current local solution); end if
         end if
         if $C_{max}(i)$ is local optimum (within its visual)
            move cockroach $i$ towards global optimum;
            construct a solution (a sequence of jobs) by
            convert rule (CR)
            if new solution is better than global
            optimum
            update global optimum (cockroach $i$ is a
            current global solution); end if
         end if
      end for $j$
   end for $i$
   //{disperse cockroaches}
   for $i$ to $n$
        move towards random solution;
        construct a solution (a sequence of jobs) by
        convert rule (CR)
         if new solution is best than global optimum
         update global optimum; end if
     end for $i$
     //{ruthless behaviour}
     randomly selected cockroach $k$
     replace cockroach $k$ by the best global cockroach
     (global optimum)
  end while
  Return the best one as sub-optimal schedule
end
```

In order to test the effectiveness and performance of FA and CSO algorithm, we applied and tested these algorithms for permutation flow shop problem to minimize the maximum completion time (*makespan*) and chose Taillard instances to our experiments [25]. Data describing the problem can be represented as a matrix of processing time with the number of rows and columns, respectively, equal to the number of machines and jobs.

Experiments were performed 10 times for each test instance, with the parameters of the algorithms. A set of flow shop problem instances included 12 Taillard instances (see Table). Among obtained results, the best, the worst and the average value of all obtained results were selected. For the best value (smallest makespan) obtained by using selected algorithms we determined the relative deviation of the found solution from the best known values C_{opt} of Taillard instances by the following formula:

$$Dev = \frac{C_{Metaheuristic} - C_{opt}}{C_{opt}} \cdot 100. \tag{13}$$

In the case of cockroach algorithm we set the following value of parameters: $w = 0.7$, *step* = 2.5, *visual* = 1. The values of firefly algorithm parameters were: $\alpha = 1$, $\beta_0 = 1$, $\gamma = 1$. The population in CA and FA algorithms is 30 individuals, the number of iterations is 1000, $x_i^{\max} = 3.2$ and $x_i^{\min} = -2.5$. In 031-056 Taillard instances we set 50 individuals.

Obtained results of firefly and cockroach algorithms are similar (see Table 2). Comparing these two algorithms we take into account the makespan value and the number of iterations required to obtain this value. For all tested instances the value of Dev are less than 10%. In most cases firefly algorithm needs less number of iterations (see *No. iteration* in Table 2). We can conclude that these algorithms are the potential solution of permutation flow shop problems with correct value of parameters.

The execution time is another indicator of the quality, but it may not be a good comparison due to different hardware and implemented language used for algorithms. Therefore, we do not take it into account.

In our experiments, for the flow shop scheduling problem, firefly and cockroach algorithms are shown to be better than the simulated annealing (SA) but worse than tabu search algorithm (TS) presented in [19] (see Table 3).

Table 2
Results obtained by the discrete-firefly and discrete-cockroach approaches

Taillard instances (jobs × machines)	Known best solution C_{opt}	Firefly algorithm (FA)			Cockroach algorithm (CSO)		
		C_{FA} best/worst	No. iteration	Dev	C_{CSO} best/worst	No. iteration	Dev
001 (20×5)	1278	1278/1324	112	0	1278/1339	246	0
002 (20×5)	1359	1366/1373	194	0.52	1366/1383	84	0.52
011 (20×10)	1582	1625/1701	270	2.72	1619/1688	413	2.33
016 (20×10)	1397	1459/1634	674	4.44	1477/1495	443	5.73
021 (20×20)	2297	2378/2405	782	3.53	2375/2407	864	3.396
026 (20×20)	2226	2292/2340	541	2.96	2334/2364	918	4.85
031 (50×5)	2724	2742/2791	213	0.66	2728/2757	65	0.15
036 (50×5)	2829	2917/2970	276	3.11	2870/2888	484	1.45
041 (50×10)	2991	3245/3308	262	8.49	3221/3229	521	7.69
046 (50×10)	3006	3183/3251	234	5.89	3192/3257	816	6.19
051 (50×20)	3771	4138/4222	251	9.73	4102/4212	864	8.78
056 (50×20)	3679	3995/4052	321	8.59	4033/4142	823	9.62

Table 3
Comparison of our selected results with results of SA and Tabu Search from literature [19]

Taillard instances (jobs × machines)	SA	TS	FA	CSO
001 (20×5)	1324	1278	1278	1278
002 (20×5)	1368	1359	1366	1366
011 (20×10)	1743	1582	1625	1619
021 (20×20)	2485	2297	2378	2375
031 (50×5)	2791	2724	2742	2728
041 (50×10)	3399	3037	3245	3221
051 (50×20)	4438	3886	4138	4102

4.2. Optimization of queueing systems. The optimization of queueing systems is very difficult, because there is not any general method to solve these problems. We also were trying to optimize some queueing systems by nature-inspired metaheuristics [16]. There are studies on the use of the genetic algorithms to solve various problems of optimization of some queueing systems, for example [26]. In the paper, we consider the optimization of system's structure, which is to find the correct number of servers m or correct number of waiting places N. We used other methods (e.g. genetic algorithm) to prepared test instances.

In the case of M/M/m/FIFO/m+N queueing system with impatient customers we seek the number of servers m and the number of waiting places N that maximize all profits. The objective function is calculated according to the formula:

$$\max \to f(m,N) = c_1\lambda(1-\pi_l) - c_2(m+N), \qquad (14)$$

where π_l is obtained using Eq. (8), parameters c_1 and c_2 denote the profit on job service and the cost of server depreciation.

For M/M/m/FIFO/N/F queueing system we seek the number of servers m that minimize the overall costs of its operation. Here we get the following objective function:

$$\min \to f(m) = r_1 m + r_2\overline{K}, \qquad (15)$$

where r_1 is the cost of server maintenance and r_2 is the cost of existing jobs in the systems.

In order to examine these algorithms, 10 runs were completed for each test instance. In all experiments, the positions of cockroaches and fireflies have been rounded to the nearest integer value using $round(x_i)$. The population was 30 individuals, each run was terminated after 50 iterations. The values of firefly algorithm parameters were: $\alpha = 0.1$, $\beta_0 = 0.1$, $\gamma = 1$. In the case of cockroach algorithm we set the following value of parameters: $w = 0.681$, *step* = 0.5, *visual* = 5. In all considered cases we assume that parameters m and N belong to range $< 1,\ 50 >$.

The conducted experiments show that the firefly and cockroach algorithms can be used to solve the problems of queueing system optimization (see Table 4). They achieved similar results for queueing systems. In the worst case, the optimum of servers' number and waiting places with use both algorithms can be found in 8 runs out of 10. These algorithms are very efficient. In most cases we can reach the maximum value of the objective function and the optimum of unknown parameters within 20 iterations (see *No. iteration* in Table 4).

We conducted many experiments, for the examples presented in the study, the use of 30 individuals was sufficient to obtain the best value.

Table 4
Results obtained by the discrete-firefly and discrete-cockroach approaches ($m, N \in < 1,\ 50 >$ for M/M/m/FIFO/m+N, $m \in < 1, 50 >$ for M/M/m/FIFO/N/F)

Queueing system	Firefly algorithm			Cockroach algorithm		
	The best obtained value and % of correct solution	Objective function	No. iteration	The best obtained value and % of correct solution	Objective function	No. iteration
Case 1: M/M/m/FIFO/m+N $\lambda = 10$, $\mu = 2$ $c_1 = c_2 = 4$, $\delta = 5$	$m = 1$, $N = 1$ 90%	24.5374	5	$m = 1$, $N = 1$ 80%	24.5374	3
Case 2: M/M/m/FIFO/m+N $\lambda = 5$, $\mu = 2$, $c_1 = 2$, $c_2 = 4$, $\delta = 1$	$m = 1$, $N = 1$ 80%	1.2669	7	$m = 1$, $N = 1$ 80%	1.2669	7
Case 3: M/M/m/FIFO/m+N, $\lambda = 100$, $\mu = 2$ $c_1 = 4$, $c_2 = 2$, $\delta = 1$	$m = 1$, $N = 1$ 90%	395.6989	4	$m = 1$, $N = 1$ 90%	395.6989	7
Case 4: M/M/m/FIFO/N/F $\lambda = 1$, $\mu = 1$ $N = 4$, $r_1 = 1$, $r_2 = 4$	$m = 3$ 100%	11.1633	17	$m = 3$ 100%	11.1633	19
Case 5: M/M/m/FIFO/N/F $\lambda = 10$, $\mu = 1$ $N = 4$, $r_1 = 1$, $r_2 = 4$	$m = 1$ 100%	16.60	12	$m = 1$ 100%	16.60	15
Case 6: M/M/m/FIFO/N/F $\lambda = 100$, $\mu = 5$ $N = 14$, $r_1 = 1$, $r_2 = 4$	$m = 1$ 100%	56.80	11	$m = 1$ 100%	56.80	19

5. Conclusions

In the study we tested the firefly and cockroach algorithms for optimization of queueing systems and solution of permutation flow shop problems. The experiments show that these methods can be used to solve these problems. The firefly and cockroach algorithms are simple to implement, but their parameters may depend on type and size of the optimized problems. It should be mentioned that the number of iterations needed to find the best value also depends on the generated initial solution. There are many potential applications of these algorithms to optimization problems. In the future we will focus on the application of these algorithms in solving other optimization problems. We would like to use swarm intelligence to solve a problem that connects scheduling and queueing jobs in a complex structure.

Acknowledgements. The authors would like to thank the referees for constructive comments and suggestions.

REFERENCES

[1] J. Kennedy and R. Eberhart, "Particle swarm optimization", *Proc. IEEE Int. Conf. on Neural Networks* 4, 1942–1948 (1995).

[2] M. Dorigo, "Optimization, learning and natural algorithms", *PhD Thesis*, Politecnico di Milano, Milano, 1992.

[3] M. Dorigo and L.M. Gambardella, "Ant colony system: a cooperative learning approach to the traveling salesman problem", *IEEE Trans. on Evolutionary Computation* 1 (1), 53–66 (1997).

[4] D.T. Pham, A. Ghanbarzadeh, E. Koc, S. Otri, S. Rahim, and M. Zaidi, "The bees algorithm – a novel tool for complex optimisation problems", in *Technical Note*, Manufacturing Engineering Centre, Cardiff University, Cardiff, 2005.

[5] B. Basturk and D. Karaboga, "An artificial bee colony (ABC) algorithm for numeric function optimization", *Proc. IEEE Swarm Intelligence Symp.* 1, CD-ROM (2006).

[6] D. Karaboga and B. Basturk, "A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm", *J. Global Optimization* 39 (3), 459–471 (2007).

[7] X.S. Yang, *Nature-Inspired Metaheuristic Algorithms*, Luniver Press, Frome, 2008.

[8] Z. Chen and H. Tang, "Cockroach swarm optimization", *II Int. Conf. on Computer Engineering and Technology* 6, 652–655 (2010).

[9] S. Łukasik and S. Żak, "Firefly algorithm for continuous constrained optimization task", *Computational Collective Intelligence. Semantic Web, Social Networks and Multiagent Systems LNCS* 5796, 97–106 (2009).

[10] Z. Chen, "A modified cockroach swarm optimization", *Energy Procedia* 11, 4–9 (2011).

[11] L. Cheng, Z. Wang, S. Yanhong, and A. Guo, "Cockroach swarm optimization algorithm for TSP", *Advanced Engineering Forum* 1, 226–229 (2011).

[12] C. Smutnicki, *Scheduling Algorithms*, Exit, Warsaw, 2002, (in Polish).

[13] B. Filipowicz, *Operational Research. Selected Methods and Algorithms. Part 1*, F.H.U. Poldex, Cracow, 1999, (in Polish).

[14] G. Bolch, S. Greiner, H. de Meer, and K.S. Trivedi, *Queueing Networks and Markov Chains. Modeling and Performance Evaluation with Computer Science Applications*, Wiley, New York, 1998.

[15] B. Filipowicz, *Modelling and Optimization of Queueing Systems. Part 1. Markov Systems*, F.H.U. Poldex, Cracow, 1999, (in Polish).

[16] J. Kwiecień and B. Filipowicz, "Firefly algorithm in optimization of queueing systems", *Bull. Pol. Ac.: Tech.* 60 (2), 363–368 (2012).

[17] R. Ruiz and C. Maroto, "A comprehensive review and evaluation of permutation flowshop heuristics", *Eur. J. Operational Research* 165 (2), 479–494 (2005).

[18] S.R. Hejazi and S. Saghafian, "Flowshop-scheduling problems with makespan criterion: a review", *Int. J. Production Research* 43 (14), 2895–2929 (2005).

[19] M. Ben-Daya and M. Al-Fawzan, "A tabu search approach for the flow shop scheduling problem", *Eur. J. Operational Research* 109 (1), 88–95 (1998).

[20] J. Grabowski and M. Wodecki, "A very fast tabu search algorithm for the permutation flow shop problem with makespan criterion", *Computers & Operations Research* 31 (11), 1891–1909 (2004).

[21] E. Nowicki and C. Smutnicki, "A fast search algorithm for the permutation flow-shop problem", *Eur. J. Operational Research* 91 (1), 160–175 (1996).

[22] Z. Lian, X. Gu, and B. Jiao, "A similar particle swarm optimization algorithm for permutation flowshop scheduling to minimize makespan", *Applied Mathematics and Computation* 175 (1), 773–785 (2006).

[23] C. Rajendran and H. Ziegler, "Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs", *Eur. J. Operational Research* 155 (2), 426–438 (2004).

[24] M.K. Sayadi, R. Ramezanian, and N. Ghaffari-Nasab, "A discrete firefly meta-heuristic with local search for makespan minimization in permutation flow shop scheduling problems", *Int. J. Industrial Engineering Computations* 1 (1), 1–10 (2010).

[25] E. Taillard, "Benchmarks for basic scheduling problems", *Eur. J. Operational Research* 64 (2), 278–285 (1993).

[26] Ch.H. Lin and J.Ch. Ke, "Optimization analysis for an infinite capacity queueing system with multiple queue-dependent servers: genetic algorithms", *Int. J. Computer Mathematics* 88 (7), 1430–1442 (2011).