International Journal of Grid and Distributed Computing
Vol. 9, No. 9 (2016), pp.379-394
http://dx.doi.org/10.14257/ijgdc.2016.9.9.32

A Coevolutionary Bacterial Foraging Model Using PSO in Job-Shop Scheduling Environments

Liang Sun, Hongwei Ge*, Limin Wang

[1.] *College of Computer Science and Technology, Dalian University of Technology, Dalian 116023, China*

[2.] *Department of Management Science and Information Engineering, Jilin University of Finance and Economics, Changchun 130117, China*

liangsun@dlut.edu.cn, hwge@dlut.edu.cn, wlm_new@163.com

Abstract

The optimization of job-shop scheduling is very important because of its theoretical and practical significance. In this paper, a computationally effective approach of combining bacterial foraging strategy with particle swarm optimization for solving the minimum makespan problem of job shop scheduling is proposed. In the artificial bacterial foraging system, a novel chemotactic model is designed to address the job shop scheduling problem and a mechanism of quorum sensing and communication are presented to improve the foraging performance. In the particle swarm system, a novel concept for the distance and velocity of a particle is presented to pave the way for the job-shop scheduling problem. The proposed coevolutionary algorithm effectively exploits the capabilities of distributed and parallel computing of swarm intelligence approaches. The algorithm is examined using a set of benchmark instances with various sizes and levels of hardness and compared with other approaches reported in some existing literatures. The computational results validate the effectiveness of the proposed algorithm.

Keywords: *Coevolutionary optimization, Bacterial foraging, Job shop scheduling.*

1. Introduction

The job shop scheduling problem (JSSP) is a very important practical problem in both fields of production management and combinatorial optimization. Efficient methods for solving the JSSP have significant effects on profitability and product quality. The JSSP has drawn the attention of researchers for the last three decades, but because scheduling problems vary widely according to specific production tasks and most of them are strongly NP-hard problems such that some test problems of moderate size are still unsolved. As a matter of fact, only small size instances of the problems can be solved within a reasonable computational time by exact optimization algorithms such as branch and bound [1, 2], and dynamic programming (DP) [3, 4], including the notorious 10×10 instance of Fisher and Thompson, which was proposed in 1963 and only solved 20 years later. Problems of dimension 15×15 are still considered to be beyond the reach of today's exact methods. By contrast, approximate and heuristic methods make a tradeoff between solution quality and computational cost. These methods mainly include dispatching priority rules [5-7], shifting bottleneck approach [8, 9], Lagrangian relaxation [10, 11], tabu search [12-14] and have made considerable achievement. In recent years, much attention has been devoted to meta-heuristics with the emergence of new techniques from the field of artificial intelligence such as genetic algorithm (GA) [15-18], simulated annealing (SA) [19-22], ant colony optimization (ACO) [23], particle swarm optimization (PSO) [24,25], artificial neural network (ANN) [26-28], bacterial foraging algorithm (BFA) [29], and so on. These meta-heuristics can be regarded as problem-independent approaches and are well suited to solve complex problems that may be difficult to solve by traditional techniques, moreover, they are capable of producing high-quality solutions with a reasonable computational effort. Among the meta-heuristic algorithms, GA has been used with increasing frequency to address scheduling problems and may not remain much room for the improvement, however, the use of the BFA and PSO for the solution of scheduling problems has been scarce, especially the BFA. Besides, many research results of the JSSP show that it is difficult to obtain a good-enough solution only by single search scheme. Motivated by these perspectives, we propose an effective cooperative intelligent algorithm for the job shop scheduling problem based on bacterial foraging strategy and particle swarm optimization in this paper. Both BFA and PSO are evolutionary computation techniques based on swarm intelligence. They exhibit implicit parallelism and contain certain redundancy and historical information of past solutions. So the proposed hybrid algorithm also effectively exploits the capabilities of distributed and parallel computing of swarm intelligence approaches.

The remainder of the paper is organized as follows. Section 2 introduces and formulates the job-shop scheduling problem. Section 3 describes a proper representation for the problem. A BFA-based scheduling algorithm and a PSO-based scheduling algorithm are proposed respectively in Section 4 and Section 5, which are followed in Section 6 showing the framework of the cooperative intelligence algorithm based on the above two sections. Section 7 reports the corresponding computational and comparative results. Finally, the conclusions are made in Section 8.

2. Job-shop Scheduling Problem

The job shop problem studied in this paper consists of scheduling a set of jobs on a set of machines with the objective of minimization the makespan, i.e., the maximum of completion times needed for processing all jobs. Each machine can handle at most one job at a time. Each job consists of a chain of operations to be processed in a specified sequence, on specified machines, and during an uninterrupted time period of given length.

Explaining the problem more specifically, let $J=\{1,2,\cdots,n\}$ denote the set of jobs, $M=\{1,2,\cdots m\}$ denote the set of machines, and $O=\{0,1,\cdots,n\times m,n\times m+1\}$ denote the set of operations to be scheduled, where 0 and $n\times m+1$ represent the dummy initial and final operations respectively. The operations are interrelated by the precedence constraints, which force each operation j to be scheduled after all predecessor operations P_j are completed. Besides, operation j can only be scheduled if the machine what it requires is idle. Further, let T_j and F_j denote the fixed processing time and the finish time of operation j respectively, let $A(t)$ be the set of operations being processed at time t, and let $e_{jm}=1$ if operation j is required to process on machine m and $e_{jm}=0$ otherwise. The conceptual model of the JSSP can be stated as [30]:

Minimize: $F_{n\times m+1}$, (1)

subject to: $F_k \le F_j - T_j, \qquad (j=1,2,\cdots n\times m+1; k\in P_j)$ (2)

$$\sum_{j\in A(t)} e_{jm} \le 1, \qquad (m\in M; t\ge 0) \tag{3}$$

$$F_j \ge 0, \qquad (j=1,2,\cdots n\times m+1) \tag{4}$$

The objective function (1) minimizes the finish time of the last operation, namely the makespan. Constraints (2) impose the precedence relations between operations. Constrains (3) represents that one machine can only process one operation at a time, and Constrains (4) force the finish times to be non-negative.

The solution to the JSSP can be represented as the operations permutation of the jobs on each machine. The total number of all possible schedules (both feasible and infeasible) is $(n!)^m$ for the problems with n jobs and m machines. Obviously, it is impossible to exhaust all the alternatives for finding the optimal solution even if the values of n and m are small. For example, for the Fisher-Thompson benchmark problem of the 10 jobs to 10 machines, it has a search space with a size at about 3.96×10^{65}. So it is necessary to restrict the search space and guide the search process. There are four types of feasible schedules in JSSP, namely inadmissible, semi-active, active and non-delay. Inadmissible schedules contain excess idle time, and they can be improved by forward-shifting operations until no excess idle time exists. Semi-active schedules contain no excess idle time, but they can be improved by shifting some operations to the front without delaying others. Active schedules contain no idle time and no operation can be finished earlier without delaying other operations. Non-delay schedules are active schedules, in which operations are placed into the schedule such that the machine idle time is minimized and no machine is kept idle if some operation can be processed. The optimal schedule is guaranteed to be an active schedule. So we only need to find the optimum solution in the set of active schedules.

3. Representation

Before solve the JSSP, we describe a proper representation for the solution of the problem, namely a scheduling, which is used in the later algorithms. In this section, an operation-based representation is adopted, which uses an unpartitioned permutation with *m*-repetitions of job numbers for the problems with *n* jobs and m machines. A job represents a set of operations that has to be scheduled on m machines. In this formulation, each job number occurs m times in the permutation. By scanning the permutation from left to right, the *k*-th occurrence of a job number refers to the *k*-th operation in the technological sequence of this job. A permutation with repetition of job numbers only expresses the order in which the operations of jobs are scheduled. For example, suppose a sequence is given as (2 1 3 4 2 4 3 2 1 2 3 3 1 1 4 4) in a 4 jobs × 4 machines problem, in which each gene stands for one operation. Each job consists of four operations, and the job number is thereby repeated four times. The fifth gene of the permutation implies the second operation of job 2 because number 2 has been repeated twice, similarly, the eighth gene represents the third operation of job 2, and so on. The prominent advantage is that the permutation is always feasible, moreover, it eliminates the deadlock schedules which are incompatible with the technological constraints and can never be finished. But it will produce redundancy in the search space and cause the search space size to expand to $(n \times m)!/(m!)^n$, that is, the mapping relation between sequence and schedule is many-to-one.

4. BFA-based Scheduling Algorithm

4.1 Standard Bacterial Foraging Algorithm

Bacterial foraging algorithm (BFA), originally developed by K. M. Passino [31], is a new natural heuristic optimization method based on foraging behavior of *E. coli* bacterium. It is also an evolutionary computation technique based on swarm intelligence. BFA has attracted broad attention in the fields of evolutionary computing, optimization and many others [32-34]. The system of the *E.coli* bacterium enables it to achieve a complex type of search and avoidance behavior. Evolution has designed this control system. It is robust and clearly very successful at meeting its goals of survival when viewed from a population perspective. In addition the control system dictates that bacterial foraging process can be subdivided into four motile behaviors namely chemotaxis, swarming, reproduction, and elimination and dispersal. Below we briefly describe each of these processes.

(i) Chemotaxis: This process simulates the movement of an E. coli cell through swimming and tumbling via flagella. Biologically an Escherichia coli bacterium can move in two different ways. It can swim for a period of time in the same direction or it may tumble, and alternate between these two modes of operation for the entire lifetime. Suppose $\theta^i(j,k,l)$ represents the position of the i th bacterium at j th chemotactic, k th reproductive and l th elimination-dispersal step. $C(i)$ is the size of the step taken in the random direction specified by the tumble. Then in computational chemotaxis the movement of the bacterium may be represented by

$$\theta^i(j+1,k,l)=\theta^i(j,k,l)+C(i)\frac{\Delta(i)}{\sqrt{\Delta^T(i)\Delta(i)}} \tag{5}$$

where Δ indicates a vector in the random direction whose elements lie in [-1,1].

(ii) Swarming: An interesting group behavior has been observed for several motile species of bacteria including E. coli and Salmonella typhimurium, where intricate and stable spatio-temporal patterns (swarms) are formed in semisolid nutrient medium [35]. A group of *E. coli* cells arrange themselves in a traveling ring by moving up the nutrient gradient when placed amidst a semisolid matrix with a single nutrient chemo-effecter. The cells when stimulated by a high level of succinate, release an attractant aspertate, which helps them to aggregate into groups and thus move as concentric patterns of swarms with high bacterial density. The cell-to-cell signaling in E. coli swarm may be represented by the following function.

$$\begin{aligned} J_{cc}(\theta,P(j,k,l)) &= \sum_{i=1}^{S} J_{cc}(\theta,\theta^i(j,k,l)) \\ &= \sum_{i=1}^{S}[-d_{attractant}\exp(-w_{attractant}\sum_{m=1}^{p}(\theta_m-\theta_m^i)^2)]+\sum_{i=1}^{S}[h_{repellant}\exp(-w_{repellant}\sum_{m=1}^{p}(\theta_m-\theta_m^i)^2)] \end{aligned} \tag{6}$$

where $J_{cc}(\theta,P(j,k,l))$ is the cost function value to be added to the actual fitness function. $P(j,k,l)$ represents the population of bacteria, namely $P(j,k,l)=\{\theta^i(j,k,l)\mid i=1,2,\cdots S\}$. S is the total number of bacteria and p is the number of parameters to be optimized which are presented in each bacterium and $\theta=[\theta_1,\theta_2,\cdots\theta_p]^T$ is a point in the p-dimensional search domain. $d_{attractant}$, $w_{attractant}$, $h_{repellant}$ and $w_{repellant}$ are different coefficients that should be chosen properly.

```
Pseudocode of bacterial foraging algorithm
Step 1. Initialize the population
Step 2. Evaluate the individual using evaluation function
Step 3. Three loops of optimization
        Inner loop: chemotactic event;
        Middle loop: duplicate event;
        Outer loop: elimination -dispersal event
Step 4. Decode the optimal individual to obtain the final solut ion
```

Figure 1. The Step Outline of the Standard BFA

(iii) Reproduction: The least healthy bacteria eventually die while each of the healthier bacteria (those yielding lower value of the objective function) asexually split into two bacteria, which are then placed in the same location. This keeps the swarm size constant.

(iv) Elimination and dispersal: Gradual or sudden changes in the local environment where a bacterium population lives may occur due to various reasons, e.g. a significant local rise of temperature may kill a group of bacteria that are currently in a region with a high concentration of nutrient gradients. Events can take place in such a fashion that all the bacteria in a region are killed or a group is dispersed into a new location. To simulate this phenomenon in BFA some bacteria are liquidated at random with a very small probability while the new replacements are randomly initialized over the search space.

The step outline of the standard BFA for solving optimization problem is shown in Figure 1.

4.2 Improved Bacterial Foraging Algorithm for Job Shop scheduling

To tackle this job shop scheduling problem, some improvements are introduced to the basic BFA. In the chemotactic step, the vector Δ of length $j \times m$ for the problem of j jobs and m machines in the random direction is randomly filled with elements of the set {0, 1}. The number of the element "1" is decided by the value of $C(i)$, namely the step size specified by the tumble, and $C(i)$ is a non-negative integer. The value of $C(i)$ is inversely proportional to the fitness of the i th bacterium. The higher the fitness a bacterium has the smaller the value. The $C(i)$ is decided by the following formulation

$$C(i) = \mathrm{int}[\frac{(f_{\max} - f_i)(r - h)}{f_{\max} - f_{\min}}] + h \tag{7}$$

where $\mathrm{int}[\cdot]$ is the truncation function, f_i is the fitness of the i th bacterium, $f_{\max}$ and $f_{\min}$ are respectively the maximal and the minimal fitness in the bacterial colony, r and h are respectively the upper bound and the lower bound of the step size, which are adjustable integers. In the process of the bacterium tumble, the $C(i)$ positions are selected from the position vector of the bacterium, in which the values of the elements in these same positions of the corresponding vector Δ are equal to 1, and then the operation of the tumble is completed by permuting the elements on these positions

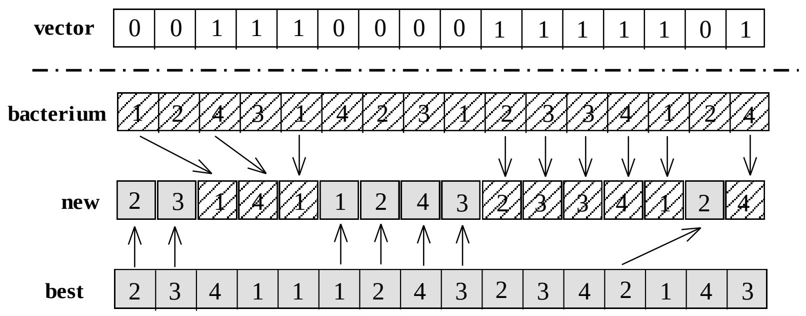

Figure 2. An Example of the QUSAC Mechanism

randomly. After the bacterium tumbles, the swim mechanism is performed. In the process, randomly select an element in the position vector of the bacterium, respectively, make it move backwards or forwards in turn and once shift one position until it moves λ times, and then 2λ new bacterial are produced. We replace the original bacterium by the new one with the highest fitness among the produced bacterial if it is better than the original.

The experimentation with complex problems reveals that the BFO algorithm possesses a poor convergence behavior. By our analysis, the emergence of the poor convergence behavior is brought by the "swarming" mechanism. On the one hand, when the bacteria quickly congregate round the local optimal position on a large scale, a high level of attractant is formed in the region. It maybe induces that the bacteria near to the global optimal position depart from the original region and close to the congregated region; on the other hand, when a large numbers of bacteria close to the global optimal position, a high level of repellent maybe dispel the current optimal bacteria away from the global optimal position. Actually, to perform a social foraging, an animal needs communication capabilities and it gains advantages to exploit essentially the sensing capabilities of the group. So a novel mechanism of quorum sensing and communication (QUSAC) is presented instead of the "swarming" mechanism. Besides, the QUSAC mechanism reduces the computation complexity largely in swarming stage.

The process of quorum sensing and communication is designed as follows. First, the bacterium with the highest fitness is recognized, and then a bacterium is selected for quorum sensing. Produce a vector of length $j \times m$ for the problem of j jobs and m machines which is randomly filled with elements of the set {0, 1}. This vector defines the order in which the elements of the newly produced bacterium vector are drawn from the best bacterium and the selected bacterium, respectively. After an element is drawn from one bacterium and deleted from the other one, it is appended to the newly produced bacterium vector. This step is repeated until both the best and the selected bacterium vector are empty and the produced bacterium contains all the elements involved. A new bacterium is created based on quorum sensing and communication and accepted into the population if its fitness is higher than the original. Figure 2 gives an example of the QUSAC mechanism.

5. PSO-based Scheduling Algorithm

The particle swarm optimization (PSO), originally developed by Kennedy and Elberhart [36], is a method for optimizing hard numerical functions on metaphor of social behaviors of flocks of birds and schools of fish. It is an evolutionary computation technique based on swarm intelligence. A swarm consists of individuals, called particles, which change their positions over time. Each particle represents a potential solution to the problem. In a PSO system, particles fly around in a multi-dimensional search space. During its flight, each particle adjusts its position according to its own experience and the experience of its neighboring particles, making use of the best position encountered by itself and its neighbors. The effect is that particles move towards the better solution areas, while still having the ability to search a wide area around the better solution areas. The performance of each particle is measured according to a pre-defined fitness function, which is related to the problem being solved. The PSO has been found to be robust and fast in solving non-linear, non-differentiable and multi-modal problems. The mathematical description and executive steps of the PSO are as follows.

Let the i th particle in a D -dimensional space be represented as $\vec{x}_i=(x_{i1},\ldots,x_{id},\ldots,x_{iD})$. The best previous position of the i th particle is recorded and represented as $\vec{p}_i=(p_{i1},\ldots,p_{id},\ldots,p_{iD})$, which gives the best fitness value and is also called $pbest$. The index of the best $pbest$ among all the particles is represented by the symbol g. The location P_g is also called $gbest$. The velocity for the i th particle is represented as $\vec{v}_i=(v_{i1},\ldots,v_{id},\ldots,v_{iD})$. The concept of the particle swarm optimization consists of changing the velocity and location of each particle towards its $pbest$ and $gbest$ locations according to Eqs. (8) and (9) at each time step:

$$v_{id}=wv_{id}+c_1r_1(p_{id}-x_{id})+c_2r_2(p_{gd}-x_{id}), \tag{8}$$

$$x_{id}=x_{id}+v_{id}, \tag{9}$$

where w is the inertia coefficient which is a constant in the interval [0, 1]; c_1 and c_2 are learning rates which are nonnegative constants; r_1 and r_2 are generated randomly in the interval [0, 1]; $v_{id} \in [-v_{\max}, v_{\max}]$, and $v_{\max}$ is a designated maximum velocity.

The conventional particle swarm optimization cannot be applied to the JSSP directly. In this section, we describe the formulations of our previous work about discrete PSO algorithm [24]. Firstly, the concept of the difference of the locations is extended. Let us present the similarity measure of two particles. Denote the i th and the j th particles in a D-dimensional space as $X_i = (x_{i1}, \ldots, x_{id}, \ldots, x_{iD})$ and $X_j = (x_{j1}, \ldots, x_{jd}, \ldots, x_{jD})$, respectively. Define functions

$$s(k) = \begin{cases} 1 & if \quad x_{ik} = x_{jk} \\ 0 & if \quad x_{ik} \neq x_{jk} \end{cases} \quad \text{and} \quad S(X_i, X_j) = \sum_{k=1}^{D} s(k) \quad , \tag{10}$$

where $S(X_i, X_j)$ is called the similarity measure between particle X_i and particle X_j. Let $f(X_i)$ denote the fitness of particle i and assume that $0 \leq f(X_i) \leq C$, $\forall i \in \{1, 2, \cdots n\}$, where n is the population size. The difference of the locations between two particles, namely "distance", is redefined by the following equation:

$$dis(X_i - X_j) = k \cdot [\alpha \cdot |\, f(X_i) - f(X_j)\,| / C + \beta \cdot (D - S(X_i, X_j)) / D] , \tag{11}$$

where D is the dimension of a particle, k is a positive integer called acceleration coefficient, and α and β are two positive weights, which can be ascertained by trial and error and satisfy the relation that the sum of α and β is equal to 1. It is obvious that $0 \leq |\, f(X_i) - f(X_j)\,| / C \leq 1$ and $0 \leq (D - S(X_i, X_j)) / D \leq 1$.

Correspondingly, the concept of the velocity is also extended. The velocity is defined as the times of "adjustment operation". The brief outline of the adjustment algorithm is as follows.

Let $X = (x_1, \ldots, x_k, \ldots, x_D)$ and $Y = (y_1, \ldots, y_k, \ldots, y_D)$ be two particles in a D-dimensional space respectively.

Step 1: Select an index k of location randomly and set an indicator $m = 0$, if $x_k = y_k = s$, where s is the number on location k, go to Step 2, else if $x_k \neq y_k$, namely $x_k = s$ and $y_k \neq s$, go to Step3.

Step 2: Scan set $X' = \{x_i \,|\, 1 \leq i < k\}$ and set $Y' = \{y_i \,|\, 1 \leq i < k\}$ from left to right, respectively, to find out the times that s appears, and denote them as $X'\_times(s) = t_x$ and $Y'\_times(s) = t_y$, respectively. If $t_x > t_y$, go to Step 4; if $t_x < t_y$, go to Step 5; if $t_x = t_y$, go to Step 6.

Step 3: Select a location index j in the particle Y randomly, which satisfies $y_j = s$ and $y_j \neq x_j$. Swap y_j and y_k, and set the indicator $m = 1$, then go to Step 2.

Step 4: Swap s which appears after location k and y_j which satisfies that $y_j \neq x_j$, $y_j \neq s$ and $j < k$ until $t_x = t_y$, then go to Step 7.

Step 5: Swap s which appears in the front of location k and y_j which satisfies that $y_j \neq x_j$, $y_j \neq s$ and $j > k$ until $t_x = t_y$, then go to Step 7.

Step 6: If $m = 0$, reselect an index k randomly; if $m = 1$, go to Step 7.

Step 7: Terminate.

Using "$\oplus$" to denote the adjustment operation, the proposed PSO algorithm could be performed by the following equations:

$$v_{id} = \text{int}[wv_{id} + c_1 r_1 dis(p_{id} - x_{id}) + c_2 r_2 dis(p_{gd} - x_{id})], \tag{12}$$

$$x_{id} = x_{id} \oplus v_{id}, \tag{13}$$

where $\text{int}[\cdot]$ represents the truncation function and $dis(\cdot)$ can be calculated using Eq.(11).

6. Coevolutionary Intelligence Algorithm based on the Proposed BFA and PSO

Based on the models of the proposed BFA and PSO, a coevolutionary intelligence algorithm (CIA) is integrated to solve the JSSP. The objective is to search a scheduling such that

$$\min(T(JM)) = \min\{\max[T(1), T(2), \cdots, T(i), \cdots, T(m)]\} \tag{14}$$

where $T(i)$ is the final completion time of machining on the *i*th machine, $T(JM)$ is the final completion time of all the jobs.

The fitness and the affinity are evaluated using the following equation

$$f_i = 100 \times opt / T_i(JM) \tag{15}$$

Table 1. Example for 3 jobs × 2 machines

	(m, t)	(m, t)
Job 1	(2, 1)	(1, 2)
Job 2	(1, 3)	(2, 2)
Job 3	(1, 1)	(2, 1)

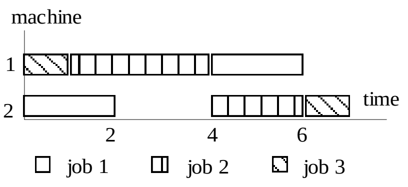

Figure 3. The Gantt chart of semi-active scheduling.

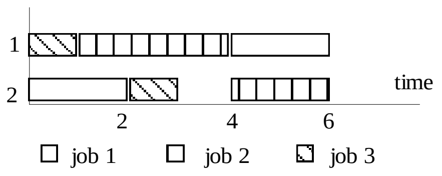

Figure 4. The Gantt chart of active scheduling.

where f_i is the fitness value of the ith bacterium in the BFA, and is also the fitness value of the *i*th particle in the PSO, opt is the theoretical optimal makespan for a given problem and the value f_i is in the interval (0, 100].

The value of the objective function *T(JM)* can be obtained by the process of decoding. The makespan is produced by the process that assigns operations to the machines at their earliest possible starting time by the technological order of each job, scanning the permutation from left to right. It should be noticed that the scheduling acquired by this way is only a semi-active. Yet, the optimal solution must be an active schedule. Then the active decoding is applied, which checks the possible blank time interval before appending an operation at the last position, and fills the first blank interval before the last operation to convert the semi-active schedule to an active one so that the makespan can be shorter. For example, consider the 3 jobs × 2 machines problem shown in Table 1. In the example, job 1 must go to machine 2 for 1 unit of time, then to machine 1 for 2 units of time, and so on. Suppose an operation-based solution is the sequence a = (1, 3, 2, 2, 1, 3) and its Gantt chart by the semi-active decoding scheme is drawn in Figure. 3, whose makespan is 7 units. Whereas the makespan is 6 units by the active decoding and the corresponding Gantt chart is shown in Figure. 4.

It is worth mentioning that the fitness is as pivotal to the guidance of the search because the chance of a solution being chosen is determined solely by its objective function value. So their values are linearly adjusted to avoid early convergence and ensure the variety of the population. The adjustment process can be summarized as follows.

Step 1. Calculate the average fitness and affinity f_{ave}, the maximal $f_{\max}$ and the minimal $f_{\min}$ in the population;

Step 2. If $f_{\min} > 2f_{ave} - f_{\max}$, go to Step 3, else go to Step 4;

Step 3. Calculate coefficients $\alpha = \frac{f_{ave}}{f_{\max} - f_{ave}}$ and $\beta = \frac{(f_{\max} - 2f_{ave})f_{ave}}{f_{\max} - f_{ave}}$. Go to Step 5;

Step 4. Calculate coefficients $\alpha = \frac{f_{ave}}{f_{ave} - f_{\min}}$ and $\beta = \frac{-f_{\min}f_{ave}}{f_{ave} - f_{\min}}$. Go to Step 5;

Step 5. Calculate the new fitness value using the equation $f' = \alpha f + \beta$.

Furthermore, the proposed algorithm stops if the best obtained solution equals to the best known makespan or if for a given number $gens$ of generations there has been no improvement of the best solution found so far, where $gens = 20$. The advantage of the proposed cooperative intelligence algorithm is that it can produce quite satisfactory solutions of the test problems in a reasonable time. The detailed pseudo-code of the improved algorithm is given below.

The cooperative intelligence algorithm for JSSP based on BFA and PSO

Step 1 Initialize parameters $D, S, N_c, N_s, N_{re}, N_{ed}, P_{ed}, cnt, w, c_1, c_2, \theta^i$. where,

D: Dimension of the search space (*j*×*m* dimension for the problem of *j* jobs and *m* machines)

S : Total number of bacteria in the population

N_c: The number of chemotactic steps

N_s: The swimming length

N_{re}: The number of reproduction steps

N_{ed}: The number of elimination-dispersal events

P_{ed}: Elimination-dispersal probability

cnt : The iterative number of the PSO algorithm
w : the inertia coefficient in PSO
c_1, c_2 : learning rates in PSO.

Step 2 Elimination-dispersal loop: $l = l + 1$.
Step 3 Reproduction loop: $k = k + 1$.
Step 4 Chemotaxis loop: $j = j + 1$.

[a] For i = 1, 2,. . ., S take a chemotactic step for bacterium i as follows.

[b] Compute the fitness value $J(i, j, k, l)$.

[c] Let $J_{last} = J(i, j, k, l)$ to save this value since we may find a better cost via a run.

[d] Update and save the best fitness value J^{best} found by far in the population and the corresponding best position θ^{best}.

[e] Compute the step size $C(i)$ using Eq. (7), generate a random vector $\Delta(i)$ with each element $\Delta_m(i), m = 1, 2, \cdots D$, a random number in set {0,1}.

[f] Perform the tumble operation in section 4.2.

[g] Perform the swim operation in section 4.2.

(i) Let $m = 0$ (counter for swim length).

(ii) While $m < Ns$ (if the bacterium has not climbed down too long),

[h] Perform the operation of the quorum sensing and communication in section 4.2.

[i] Go to next bacterium $(i+1)$ if $i \neq S$ (i.e., go to [b] to process the next bacterium).

[j] Perform the PSO operation according to Eqs. (12-13) in section 5 for a given iterative number cnt.

Step 5 If $j < N_c$, go to step 4. In this case, continue chemotaxis, since the life of the bacteria is not over.
Step 6 Reproduction:

[a] For the given k, and for each $i = 1, 2, \cdots S$, let J^i_{health} be the health of the i th bacterium. Sort bacteria in descending order of cost J_{health} (lower cost means lower health).

[b] The S_r bacteria with the lowest J_{health} values die and the other S_r bacteria with the best values split and the copies that are made are placed at the same location as their parent.

Step 7 If $k < N_{re}$, go to step 3. In this case, we have not reached the number of specified reproduction steps, so we start the next generation of the chemotactic loop.

Step 8 Elimination-dispersal: For $i = 1, 2, \cdots S$ with probability P_{ed}, eliminate and disperse each bacterium, and this results in keeps the number of bacteria in the population constant. To do this, if a bacterium is eliminated, simply disperse another one to a random location on the optimization domain. If $l < N_{ed}$, then go to step 2; otherwise end.

7. Numerical Simulation Results and Comparisons

Table 2. CIA Parameters

Parameters	value
Number of bacteria and particles (S)	80
Chemotactic steps (N_c)	100
Swim steps (N_s)	5
Reproduction steps (N_{re})	3

Elimination and dispersal steps (N_{ed})	<<500
Probability of elimination (P_{ed})	0.15
Iterative number of the PSO (cnt)	10
Inertia coefficient in PSO (w)	0.5
learning rates in PSO (c_1)	1.8
learning rates in PSO (c_2)	1.0

The performance of the proposed cooperative intelligence algorithm (CIA) for the JSSP is examined by using some test problems taken from the OR-Library [37]. We consider 43 instances from two classes of standard JSSP test problems: instances FT06, FT10, FT20 designed by Fisher and Thompson (1963), and instances LA01 to LA40 designed by Lawrence (1984). Numerical experiments are performed in C++ language on a PC with Pentium IV 2.93 GHz processor and 2GB memory. The numerical results are compared with those reported in some existing literatures using other approaches [30, 38-42], including some heuristic and meta-heuristic algorithms. The selection of parameter settings in CIA is listed in Table 2. The proposed algorithm stops if the best

Table 3. Comparisons of the Results between CIA and Other Approaches

Instance	Size	BKS	HEA	RD(%)	Lin[38]	Goncalves [30]			Ombuki[39]	Coello[40] [42]	Aiex[41]	Binato[42]
						Param.A	HGA	Active	LSGA	AIS	GP+PR	MGA
Ft06	6×6	55	**55**	0.00	55	55	55	55	-	-	55	55
Ft10	10×10	930	**930**	0.00	930	930	951	945	-	941	930	930
Ft20	20×5	1165	**1165**	0.00	1165	1165	1178	1173	-	-	1165	1165
La01	10×5	666	**666**	0.00	666	666	666	666	-	666	666	666
La02	10×5	655	**655**	0.00	655	655	665	655	-	655	655	-
La03	10×5	597	**597**	0.00	597	597	604	603	-	597	597	-
La04	10×5	590	**590**	0.00	590	590	590	598	-	590	590	-
La05	10×5	593	**593**	0.00	593	593	593	593	-	593	593	-
La06	15×5	926	**926**	0.00	926	926	926	926	-	926	926	926
La07	15×5	890	**890**	0.00	890	890	890	890	-	890	890	-
La08	15×5	863	**863**	0.00	863	863	863	863	-	863	863	-
La09	15×5	951	**951**	0.00	951	951	951	951	-	951	951	-
La10	15×5	958	**958**	0.00	958	958	958	958	-	958	958	-
La11	20×5	1222	**1222**	0.00	1222	1222	1222	1222	-	-	1222	1222
La12	20×5	1039	**1039**	0.00	1039	1039	1039	1039	-	-	1039	-
La13	20×5	1150	**1150**	0.00	1150	1150	1150	1150	-	-	1150	-
La14	20×5	1292	**1292**	0.00	1292	1292	1292	1292	-	-	1292	-
La15	20×5	1207	**1207**	0.00	1207	1207	1207	1207	-	-	1207	-
La16	10×10	945	**945**	0.00	945	945	973	947	959	945	945	945
La17	10×10	784	**784**	0.00	784	784	792	784	792	785	784	-
La18	10×10	848	**848**	0.00	848	848	855	848	857	848	848	-
La19	10×10	842	**842**	0.00	842							
La20	10×10	902	**902**	0.00	902							
La21	15×10	1046	**1046**	0.00	1046	1046	1079	1074	1114	-	1057	1058
La22	15×10	927	**932**	0.54	932	935	950	962	989	-	927	-
La23	15×10	1032	**1032**	0.00	1032	1032	1032	1032	1035	-	1032	-
La24	15×10	935	**939**	0.43	941							
La25	15×10	977	**977**	0.00	977							
La26	20×10	1218	**1218**	0.00	1218	1218	1218	1237	1307	-	1218	1218
La27	20×10	1235	**1239**	0.32	1239	1256	1282	1280	1350	-	1269	-
La28	20×10	1216	**1216**	0.00	1216	1232	1250	1250	1312	1277	1225	-
La29	20×10	1152	**1179**	2.34	1173	1196	1206	1226	1311	1248	1203	-
La30	20×10	1355	**1355**	0.00	1355	1355	1355	1355	1451	-	1355	-
La31	30×10	1784	**1784**	0.00	1784	1784	1784	1784	1784	-	1784	1784
La32	30×10	1850	**1850**	0.00	1850	1850	1850	1850	1850	-	1850	-

La33	30×10	1719	**1719**	0.00	1719	1719	1719	1719	1745	-	1719	-
La34	30×10	1721	**1721**	0.00	1721	1721	1721	1721	1784	-	1721	-
La35	30×10	1888	**1888**	0.00	1888	1888	1888	1888	1958	1903	1888	-
La36	15×15	1268	**1274**	0.47	1278	1279	1303	1313	1358	1323	1287	1291
La37	15×15	1397	**1411**	1.00	1411	1408	1437	1444	1517	-	1410	-
La38	15×15	1196	**1213**	1.42	1208	1219	1252	1228	1362	1274	1218	-
La39	15×15	1233	**1233**	0.00	1233	1246	1250	1265	1391	1270	1248	-
La40	15×15	1222	**1229**	0.57	1225	1241	1252	1246	1323	1258	1244	-

obtained solution equals to the best known makespan or if for a given number $gens$ (outer loop in CIA) of generations there has been no improvement of the best solution found so far, where $gens = 20$.

Table 3 summarizes the results of the experiments. The contents of the table include the name of each test problem (Instance), the scale of the problem (Size), the value of

Table 4. Average Experimental Results using the CIA

Instance	Size	BKS	Mean	MRD (%)	SD	Best	BRD (%)
Ft06	6×6	55	55	0.00	0.00	55	0.00
Ft10	10×10	930	932	0.22	2.19	930	0.00
Ft20	20×5	1165	1170	0.43	4.52	1165	0.00
La01	10×5	666	666	0.00	0.00	666	0.00
La06	15×5	926	926	0.00	0.00	926	0.00
La11	20×5	1222	1222	0.00	0.00	1222	0.00
La16	10×10	945	945	0.00	0.00	945	0.00
La21	15×10	1046	1050	0.38	4.20	1046	0.00

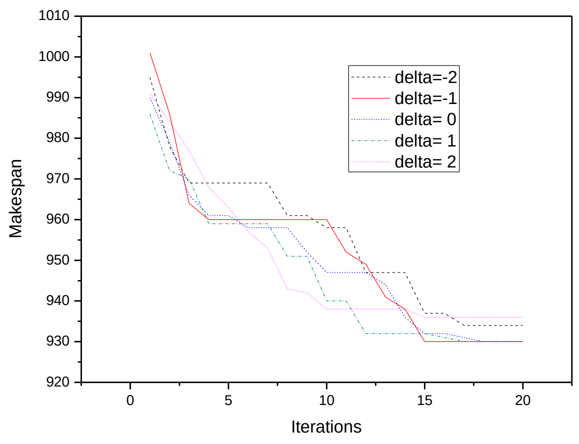

Figure 5. Convergence Curves for Different Values of the Parameter d.

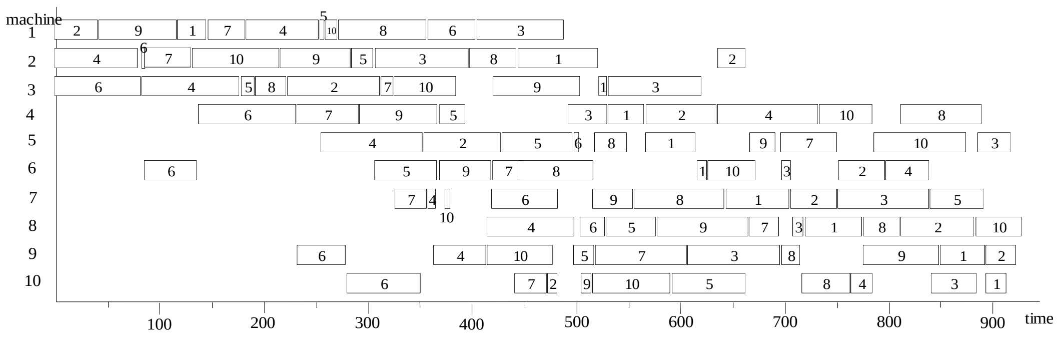

Figure 6. The Gantt Chart of an Optimal Schedule for FT10

the best known solution for each problem (BKS), the value of the best solution found by using the proposed algorithm (HIA), the number of running generations (Iterations), the percentage of the deviation with respect to the best known solution (RD%), and the best results reported in other literatures. It is worth mentioning that the makespan listed in the table is the best obtained from five executions by inching the defined fitness function (15), which is changed according to the following equation:

$$f_i = 100 \times (opt \pm \delta) / T_i\,(JM) \tag{16}$$

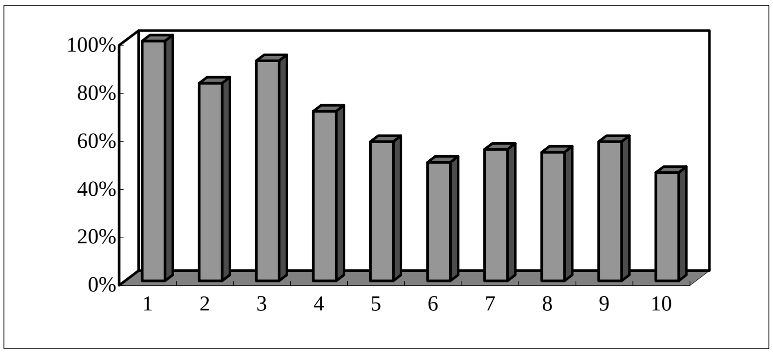

Figure 7. The Column Chart of Percentage of Utilization

where δ is the tuning parameter and its value is taken as 0, 1 and 2 respectively. The proposed algorithm produces quite satisfactory solutions in reasonable amount of iterations by tuning only the parameter δ.

From the table it can be seen that the proposed algorithm is able to find the best known solution for 35 instances, i.e. in about 81% of the instances, and the deviation of the minimum found makespan from the best known solution is only on average 0.16%. The proposed algorithm yields a significant improvement in solution quality with respect to other algorithms. The superior results indicate the successful incorporation of the improved BFA and PSO, which facilitates the escape from local minimum points and increase the possibility of finding a better solution. So it could be concluded that the proposed cooperative intelligence algorithm solves the JSSP fairly efficiently.

As above-mentioned, the algorithm is performed five times for every test problem by taking the parameter δ as 0, 1 and 2 respectively. Table 4 lists the mean solutions (Mean), the relative deviation of the mean solution (MRD), the standard deviation of the solutions (SD), the best solution (Best), and the relative deviation of the best solution (BRD). The MRD is commonly zero for small size problem, and is not more than 1.5% for most other problems.

To illustrate the simulated results more intuitively, the notorious FT10 problem is specially described as an example, in which there are numerous local optima so that the problem is challenging enough. To investigate the effect of δ-tuning on the solution quality and convergence, Figure 5 shows a group of convergence curves for different values of the parameter δ. Figure 6 shows the Gantt chart of a best solution.

Define the utilization percentage of the *i*th machine

$$U_i = \frac{P_i}{I_i + P_i} \times 100\% , \tag{17}$$

where P_i is the total processing time of machine *i*, I_i is the total idle time of machine *i* before the last operation in the machine is processed. Figure 7 shows the percentage of utilization of each machine.

8. Conclusions

A coevolutionary intelligence algorithm combing an improved BFA and PSO is proposed to solve job shop problems with minimizing the makespan. In the artificial bacterial foraging system, a novel chemotactic model is designed to address the job shop scheduling problem and a mechanism of quorum sensing and communication are presented to improve the foraging performance. In the particle swarm system, a novel concept for the distance and velocity of a particle is presented for the job shop scheduling problem. The proposed coevolutionary algorithm effectively exploits the capabilities of distributed and parallel computing of swarm intelligence approaches. The algorithm is tested on a set of 43 standard instances taken from the OR-Library. Computational results are compared with those obtained using other existing approaches and the proposed approach yields significant improvement in solution quality. The superior results indicate the successful incorporation of the two improved swarm intelligence approaches. The investigation on further study for the theoretical research as well as the performance of the technique is in progress. One of the important issues is to extend the proposed algorithm in order that it could be applied to more practical and integrated manufacturing problems such as dynamic arrivals, machine breakdown, or other factors that affect job status over time.

Acknowledgements

The authors would like to thank the support of the National Natural Science Foundation of China (61572104, 61103146, 61402076, 61502072, 61572225), Startup Fund for the Doctoral Program of Liaoning Province (20141023), the Fundamental Research Funds for the Central Universities (DUT15QY26, DUT15RC(3)088), and the project of the Key Laboratory of Symbolic Computation and Knowledge Engineering in Jilin University (93K172016K11).

References

[1] Nessah, R.and Kacem, I. (2012). Branch-and-bound method for minimizing the weighted completion time scheduling problem on a single machine with release dates, *Computers & Operations Research*, 39(3), pp.471-478.

[2] Akkan, C. and Karabati, S. (2004). The two-machine flowshop total completion time problem: Improved lower bounds and a branch-and-bound algorithm, *European Journal of Operational Research*, 159(2), pp.420-429.

[3] Jiang,W., Chen, K., Zhong, X.Q., Wang, C.E. and Zhu, C.A. (2008). Resource constrained stochastic job scheduling based on dynamic programming, *Computer Engineering*, 34(16), pp.19-21.

[4] Potts, C.N. and Van Wassonhove, L.N. (1987). Dynamic programming and decomposition approaches for the single machine total tardiness problem, *European Journal of Operational Research*, 32, pp. 405-414.

[5] Canbolat, Y.B. and Gundogar, E. (2004). Fuzzy priority rule for job shop scheduling, *Journal of intelligent manufacturing*, 15(4), pp. 527-533.

[6] Klein, R. (2000). Bidirectional planning: improving priority rule-based heuristics for scheduling resource-constrained projects, *European Journal of Operational Research*, 127(3), pp. 619-638.

[7] Weng, M.X. and Ren, H.Y. (2006). An efficient priority rule for scheduling job shops to minimize mean tardiness, *IIE Transcations*, 38(9), pp.789-795.

[8] Topaloglu, S. and Kilincli, G. (2009). A modified shifting bottleneck heuristic for the reentrant job shop scheduling problem with makespan minimization, *International Journal of Advanced Manufacturing Technology*, 44(7-8), pp.781-794.

[9] Liu, S.Q. and Kozan, E. (2012). A hybrid shifting bottleneck procedure algorithm for the parallel-machine job-shop scheduling problem, *Journal of the Operational Research Society*, 63(2), pp.168-182.

[10] Chen, H.X. and Luh, P.B. (2003). An alternative framework to Lagrangian relaxation approach for job shop scheduling, *European Journal of Operational Research*, 149(3), pp. 499-512.

[11] Jeong, I.J. and Yim, S.B. (2009). A job shop distributed scheduling based on Lagrangian relaxation to minimise total completion time, *International Journal of Production Research*, 47(24), pp. 6783-6805.

[12] Geyik, F. and Cedimoglu, I.H. (2004). The strategies and parameters of tabu search for job-shop scheduling, *Journal of intelligent manufacturing*, 15(4), pp. 439-448.

[13] Watson, J.P., Beck, J.C. and Howe, A.E. (2003). Problem difficulty for tabu search in job-shop scheduling, *Artificial intelligence*, 143(2), pp. 189-217.

[14] Li, J.Q., Pan, Q.K., Suganthan, P.N. and Chua, T.J. (2011). A hybrid tabu search algorithm with an efficient neighborhood structure for the flexible job shop scheduling problem, *International Journal of Advanced Manufacturing Technology,* 52(5-8), pp.683-697.

[15] Kachitvichyanukul, V. and Sitthitham, S. (2011). A two-stage genetic algorithm for multi-objective job shop scheduling problems, *Journal of Intelligent Manufacturing*, 22(3), pp. 355-365.

[16] Yusof, R., Khalid, M., Hui, G.T., Yusof, S.M. and Othman, M.F. (2011). Solving job shop scheduling problem using a hybrid parallel micro genetic algorithm, *Applied Soft Computing*, 11(8), pp.5782-5792.

[17] Gang, X. and Wu, Z.M. (2004). Deadlock-free scheduling strategy for automated production cell, *IEEE Transactions on Systems Man and Cybernetics Part A-Systems and Humans*, 34(1), pp.113-122.

[18] Fayad, C. and Petrovic, S. (2005). A fuzzy genetic algorithm for real-world job shop scheduling, *Lecture Notes in Computer Science*, 3533, pp.524-533.

[19] Zhang, R. and Wu, C. (2011) A simulated annealing algorithm based on block properties for the job shop scheduling problem with total weighted tardiness objective, *Computers & Operations Research*, 38(5), pp. 854-867.

[20] Zhang, R. and Wu, C. (2010) A hybrid immune simulated annealing algorithm for the job shop scheduling problem, *Applied Soft Computing*, 10(1), pp. 79-89.

[21] Kolonko, M. (1999). Some new results on simulated annealing applied to the job shop scheduling problem, *European Journal of Operational Research*, 113(1), pp. 123-136.

[22] Suresh, R.K. and Mohanasundaram, K.M. (2006). Pareto archived simulated annealing for job shop scheduling with multiple objectives, *International Journal of Advanced Manufacturing Technology*, 29(1-2), pp. 184-196.

[23] Xing, L.N., Chen, Y.W., Wang, P., Zhao, Q.S. and Xiong, J. (2010) Knowledge-Based Ant Colony Optimization for Flexible Job Shop Scheduling Problems, *Applied Soft Computing*, 10(3), pp. 888-896.

[24] Ge, H.W., Sun, L., Liang, Y.C. and Qian F. (2008). An effective PSO-and-AIS-based hybrid intelligent algorithm for job-shop scheduling. *IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans*, 38(2), pp.358-368.

[25] Xia, W.J. and Wu, Z.M. (2006). A hybrid particle swarm optimization approach for the job-shop scheduling problem, *International Journal of Advanced Manufacturing Technology*, 29(3-4), pp.360-366.

[26] Yang, S.X., Wang, D.W., Chai, T.Y., and Kendall, G. (2010). An improved constraint satisfaction adaptive neural network for job-shop scheduling, *Journal of Scheduling*, 13(1), pp. 17-38.

[27] Yahyaoui, A., Fnaiech, N. and Fnaiech, F. (2011). A Suitable Initialization Procedure for Speeding a Neural Network Job-Shop Scheduling, *IEEE Transactions on Industrial Electronics*, 58(3), pp. 1052-1060.

[28] Yang, S. and Wang, D. (2000). Constraint satisfaction adaptive neural network and heuristics combined approaches for generalized job-shop scheduling, *IEEE Transactions on Neural Networks*, 11(2), pp.474-486.

[29] Wu, C.G., Zhang, N., Jiang, J.Q. and Liang, Y.C. (2007). Improved bacterial foraging algorithms and their applications to job shop scheduling problems, Lecture Notes in Computer Science, 4431, pp. 562-569.

[30] Goncalves, J.F., Mendes, J.J.D.M. and Resende, M.G.C. (2005). A hybrid genetic algorithm for the job shop scheduling problem, *European Journal of Operational Research*, 167(1), pp. 77-95.

[31] Passino, K.M. (2002). Biomimicry of bacterial foraging for distributed optimization and control, *IEEE Control Systems Magazine*, 22(3), pp.52-67.

[32] Bhushan, B. and Singh, M. (2011). Adaptive control of DC motor using bacterial foraging algorithm, *Applied Soft Computing*, 11(8), pp.4913-4920.

[33] Kumar, M.S. and Renuga, P. (2011). Application of Bacterial Foraging Algorithm for Optimal Location of FACTS Devices with Multi-Objective Functions, *International Review of Electrical Engineering*, 6(4), pp.1905-1915.

[34] Sathya, P.D. and Kayalvizhi, R. (2011). Optimal segmentation of brain MRI based on adaptive bacterial foraging algorithm, *Neurocomputing*, 74(14-15), 2299-2313.

[35] Budrene, E. and Berg, H. (1995). Dynamics of formation of symmetrical patterns by chemotactic bacteria, *Nature*, 376(6535), pp.49–53.

[36] Kennedy, J. and Eberhart, R. (1995). Particle Swarm Optimization, *Proceedings of the IEEE International Conference on Neural Networks*, Perth, Australia, IEEE Service Center, Piscataway, NJ, 4, pp.1942–1948.

[37] Beasley, J.E. (1990). OR-Library: Distributing Test Problems by Electronic Mail, *Journal of the Operations Research Society*, 41(11), pp. 1069-1072.

[38] Lin, T.L., Horng, S.J., Kao, T.W., Chen, Y.H., Run, R.S., Chen, R.J., Lai, J.L. and Kuo, I.H. (2010). An efficient job-shop scheduling algorithm based on particle swarm optimization, Expert Systems with Applications, 37(3), pp. 2629-2636.

[39] Ombuki, B.M. and Ventresca, M. (2004). Local search genetic algorithms for the job shop scheduling problem, *Applied Intelligence*, 21(1), pp. 99-109.

[40] Coello, C.A.C., Rivera, D.C. and Cortes, N.C. (2003). Use of an artificial immune system for job shop scheduling, *Lecture Notes in Computer Science*, 2787, pp.1-10.

[41] Aiex, R.M. Binato, S. and Resende, M.G.C. (2003). Parallel GRASP with path-relinking for job shop scheduling, *Parallel Computing*, 29(4), pp. 393-430.

[42] Binato, S., Hery, W.J., Loewenstern, D.M. and Resende, M.G.C. (2001). A GRASP for Job Shop Scheduling, Essays and Surveys in Metaheuristics, Kluwer Academic Publishers, Boston, pp. 59-80.