Knowledge-Based Systems 172 (2019) 104–122

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

Bacterial foraging optimization algorithm in robotic cells with sequence-dependent setup times

Arindam Majumder [a], Dipak Laha [a], P.N. Suganthan [b,*]

[a] Department of Mechanical Engineering, Jadavpur University, Kolkata, India
[b] School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore

ARTICLE INFO

Article history:
Received 12 November 2017
Received in revised form 28 December 2018
Accepted 14 February 2019
Available online 26 February 2019

Keywords:
2-machine robotic cell
Sequencing of parts
Sequencing of robot moves
Cycle time
Minimal part set sequence
Bacterial foraging algorithm

ABSTRACT

In this paper, we propose an improved discrete bacterial foraging algorithm to determine the optimal sequence of parts and robot moves in order to minimize the cycle time for the 2-machine robotic cell scheduling problem with sequence-dependent setup times. We present a method to convert the solutions from continuous to discrete form. In addition, two neighborhood search techniques are employed to updating the positions of each bacterium during chemotaxis and elimination–dispersal operations in order to accelerate the search procedure and to improve the solution. Moreover, a multi-objective optimization algorithm based on NSGA-II combined with the response surface methodology and the desirability technique is applied to tune the parameters as well as to enhance the convergence speed of the proposed algorithm. Finally, a design of experiment based on central composite design is used to determine the optimal settings of the operating parameters of the proposed algorithm. The results of the computational experimentation with a large number of randomly generated test problems demonstrate that the proposed method is relatively more effective and efficient than the state-of-the-art algorithms in minimizing the cycle time in the robotic cell scheduling.

1. Introduction

The robotic cell scheduling (RCS) problem deals with determining the optimal sequence of parts and robot moves to minimize cycle time in a robotic cell. A robotic cell consists of an input station, a series of CNC machines, an output station and one or more robots that transport the materials within the cell. Typical applications of robotic cells in advanced manufacturing systems include semiconductor manufacturing, automobiles, engine block manufacturing, and textile mill industries [1].

This paper considers the 2-machine RCS problem with sequence dependent setup times where a family of similar parts is processed on the CNC machines for repetitive production. The objective of this study is to determine the optimal sequence of robot moves and parts to minimize cycle time, or equivalently, to maximize the throughput time. A cycle is defined as the sequence of robot moves for repetitive production to transfer a set of parts for processing through the machines and the total time taken by the robot to complete one cycle is known as the cycle time.

Following the standard notation for classifying the RCS problems in scheduling literature [2], we denote this problem as $RF_2\,|(free, A, MP, cycle-1)|\,\mu$, where RF_2 represents a 2-machine robotic cell with one simple gripper robot, $(free, A, MP, cycle-1)$ indicates free-pickup criterion, additive travel time, and multiple part type production with one unit per cycle and μ is the throughput time. Dawande et al. [2] and Kamoun et al. [3] provided a detailed survey of various classification schemes, optimization methods, applications and developments related to various RCS problems [4–6]. Nejad et al. [4] formulated a mathematical model based on traveling salesman problem's network flow modeling approach for the scheduling of flexible robotic cells. The robotic cell considered in their study consists of an input and an output buffer, a single gripper robot, and m identical parallel machines. They proposed a simulated annealing to solve large-sized RCS problem. Al-Ahmari [5] utilized Timed Petri Nets (TPNs) for modeling the RCS problem with single gripper robot. A standard commercial solver such as LINGO is used to solve this TPNs model with the objective of minimizing the cycle time. Elmi and Topaloglu [7] implemented ant colony optimization to solve the robotic cell with multiple robots. Liu and Kozan [6] considered a real-life RCS problem with multiple stationary robots and one mobile robot. They proposed a new scheduling problem, called blocking job shop scheduling problem with robotic transportation. They applied a hybrid Tabu Search and threshold accepting metaheuristic algorithm to solve this RCS problem with the objective of maximizing the throughput time.

* Corresponding author.
E-mail addresses: arindam2012@gmail.com (A. Majumder), dipaklaha_jume@yahoo.com (D. Laha), epnsugan@ntu.edu.sg (P.N. Suganthan).

https://doi.org/10.1016/j.knosys.2019.02.016
0950-7051/

Since the RCS problem belongs to NP-hard in strong sense [8], it is computationally infeasible to solve medium and large sized RCS problems using exact optimization techniques due to their exponential time complexity. On the other hand, heuristic and metaheuristic approaches due to their polynomial time complexity are preferred as suitable optimization techniques to solve these problems.

This paper presents an improved discrete bacterial foraging algorithm (IBFA) to solve the $RF_2\,|(free, A, MP, cycle-1)|\,\mu$ problem. First, the position of a bacterium in continuous variables is converted into the discrete variables. Next, to accelerate the search procedure for a better solution, the position of each bacterium during chemotaxis and elimination–dispersal in the proposed algorithm is updated using two neighborhood search techniques, namely, cyclic shift and pair-wise interchange methods. A multi-objective optimization algorithm based on NSGA-II combined with response surface methodology and desirability technique is applied to tune the parameters as well as to enhance the convergence speed of the algorithm. The proposed IBFA is compared with the best-known metaheuristic algorithms: GA1 [9], SA [10], discrete cuckoo search (DCS) algorithm [11], and two variants of the discrete BFA, basic-BFA [12] and chaotic local search-based (CLS-BFA) [12]. The computational results with many randomly generated problem instances, especially on large-sized datasets demonstrate that the proposed algorithm produces better solutions than these algorithms.

The remaining of the article is organized as follows: Section 2 describes the problem and provides the review of the literature. Section 3 presents a brief discussion of the basic BFA and describes the proposed IBFA algorithm. Section 4 provides the analysis of the design of experiment to tune the parameters of the algorithm. Section 5 presents a non-parametric performance measure based on two-tailed Wilcoxon Signed Rank test. Section 6 discusses the comparative experimental results, and finally, the conclusions are drawn in Section 7.

2. Problem description and literature review

We now describe the problem considered in this paper and briefly review the literature for solving the RCS problem.

2.1. Problem description

Consider the following scenario: At time 0, each of different part types, say, k with a total demand of R parts where, $R = p_1 + p_2 + \cdots + p_k$ is to be processed on two CNC machines in a robotic cell. The robotic cell consists of single-gripper robot, one input drive, one output drive and two processing machines. We assume that the setup times are dependent of the processing sequence of the parts and there is no buffer storage between the machines for this problem. Now, since it is computationally infeasible to determine the optimal sequence of k different parts with a total demand of R parts ($R = d_1 + d_2 + d_3 + \cdots + d_k$), where, d_k is the demand of kth part. We consider r short cycles and these cycles are repeated to produce R parts. N is given as $R = rn$, where, $n = n_1 + n_2 + \cdots + n_k$, n is known as the minimal part set (MPS), n_k is the minimum number of parts of type k in one MPS and r is the greatest common divisor of $n_1, n_2, \ldots, n_k$ demands.

For example, five different parts needs processing though the robotic cell and their corresponding demands are 200, 100, 150, 170 and 180, respectively. Here, the total number of parts, $R = 800$. Thus, it is computationally infeasible to determine the optimal sequence of all the 800 parts. In order to reduce the complexity of this optimization problem, we divide the total number of parts so that it is exactly multiple of a set of parts, known as minimal part set (MPS). Here, the greatest common divisor of all the demands of the parts, $r = 10$. Therefore, the number of parts in the MPS, $n = 80$. Thus, we can determine the optimal cycle time of all the parts by finding the optimal MPS sequence of 80 parts and repeating the obtained optimal MPS sequence by 10 times.

The problem is to determine the optimal sequence of robot moves as well as the MPS sequence of n parts of k types in the MPS set to minimize cycle time. Here, MPS sequence refers to a solution that represents a sequence among the $^{n!}/_{\prod_{s=1}^{k}(n_s!)}$ total number of feasible sequences over MPS. We consider two machines, say, m_1 and $m_2$along with one input drive (I) and one output drive (O) for the 2-machine RCS and the robot travels following two possible sequences RS_1 and RS_2. We assume each of these sequences starts from the loading of ith part on m_2. Each robot sequence consists a set of activities that are performed in order. We assume that the robot travels following two possible sequences RS1 and RS2. The difference between RS1 and RS2 is that there is no waiting time in case of RS1, whereas, for RS2, we consider the waiting time. Following are the activities of RS1 and RS2.

RS1

- The robot waits at m_2 till the processing of ith part has been completed
- Unloads the ith part from m_2
- Moves to O, drops the part at O
- Moves freely to I, picks up (i+1)th part
- Moves to m_1, loads the (i+1)th part on m_1
- Waits at m_1 until the (i+1)th part has been processed at m_1
- Unloads the (i+1)th part from m_1
- Moves to m_2
- Loads the (i+1)th part on m_2

RS2:

- The robot moves freely to I after loading ith part on m_2
- Picks up (i+1)th part from I, moves to m_1
- Loads (i+1)th part on m_1
- Moves freely to m_2
- Waits at m_2 (if necessary) until ith part is being processed at m_2
- Unloads the ith part from m_2
- Moves to O, drops the ith part at O
- Moves freely to m_1, waits at m_1 (if necessary) until the (i+1)th part is being processed at m_1
- Unloads the (i+1)th part from m_1
- Moves to m_2
- Loads (i+1)th part on m_2

Let the time interval between the loading of the part in the ith position of the MPS sequence on m_2 and the loading of the part in $(i+1)$th position of that sequence on m_2 for the robot sequence RS_1 and RS_2 be $pct_1(\sigma_i, \sigma_{i+1})$ and $pct_2(\sigma_i, \sigma_{i+1})$, respectively. The cycle time (CT) is defined as the sum of the partial cycle times (pct) between two consecutive parts in the ith and $(i+1)$th position of the sequence and it can be written as:

$$\boldsymbol{CT} = \sum_{\boldsymbol{i}=1}^{\boldsymbol{n}} (\boldsymbol{pct}\,(\boldsymbol{\sigma_i}, \boldsymbol{\sigma_{i+1}})) \tag{1}$$

If MPS sequence is considered, the optimal CT^* is given as:

$$\boldsymbol{CT^*} = \sum_{\boldsymbol{i}=1}^{\boldsymbol{n}} \boldsymbol{min}(\boldsymbol{pct}_1\,(\boldsymbol{\sigma_i}, \boldsymbol{\sigma_{i+1}})\,, \boldsymbol{pct}_2\,(\boldsymbol{\sigma_i}, \boldsymbol{\sigma_{i+1}})) \tag{2}$$

2.2. Literature review

There have been numerous studies on the application of various optimization techniques to the cyclic RCS problems. Regarding exact optimization techniques, researchers applied mixed integer programming [13–16] and branch and bound algorithms [17–19] to solve cyclic RCS problems. However, these methods because of their exponential time complexity can solve only small-sized problems.

On the other hand, polynomial algorithms like heuristics and metaheuristics are frequently applied to solve large-sized problems. Lei et al. [20], Yih [21], Batur et al. [22], and Carlier et al. [9] proposed some heuristics, but the solutions produced by these heuristics are not good enough as compared to the corresponding optimal solutions, especially for the large-sized RCS problems.

Alternatively, metaheuristics have been emerged to produce good near-optimal solutions, especially for large-sized scheduling problems requiring reasonable computational effort. Researchers proposed different metaheuristics to solve large-scale optimization problems including RCS problems. Some noteworthy metaheuristics comprise SA [8], GA [23–25], particle swarm optimization [26,27], differential evolution [28–30], DCS [11], and ant colony optimization [25].

Recently, a new metaheuristic, bacterial foraging algorithm (BFA) developed by Passino [31] draws the attention of various researchers due to its superior performance over some metaheuristics [32–35] for solving real-life optimization problems. The BFA is a computational intelligence technique. It has the ability to generate frequently good near-optimal solutions on non-linear data and large-sized complex problems. The other advantages of this algorithm include less computational effort and global convergence. Moreover, the algorithm has already shown its superiority over many evolutionary algorithms, namely, differential evolution [32], genetic algorithms [33,36], and other swarm-inspired algorithms, like, particle swarm optimization algorithm [34,35]. It has been successfully applied in a wide variety of engineering and management problems such as rapid prototyping [37], scheduling [38,39], image processing [40,41], and short-circuit analysis of induction motor [42]. A considerable group of authors such as Wu et al. [43], Zhao et al. [12], Nouri and Hong [38], Liu et al. [39], Chana [44], Devi and Geethanjali [45], Liu et al. [46,47,48], Atasagun and Kara [49], and Prakash and Vidyarthi [50] used the discrete BFA in various scheduling problems including the cellular manufacturing system, grid computing and line balancing.

Wu et al. [43] implemented individual-based search operator for increasing the effectiveness of the BFA in job shop scheduling. They have shown that the BFA produces better solutions compared to the standard BFA and some well-known algorithms. Zhao et al. [12] introduced a differential evolution operator and a chaotic local search operator with the BFA to increase the convergence speed and searching capability for solving permutation flowshop scheduling problems.

Nouri and Hong [38] proposed a new version of BFA for solving cell formation problem. The results of their algorithm is compared with some noteworthy heuristics and metaheuristics in terms of number of exceptional elements and voids in cells. Later, they used the BFA to solve problems related to cell formation with cell load variations [51–53]. Liu et al. [46] proposed two variants of discrete BFA, namely, BFA1 and BFA2 to form cells and to sequence task in cellular manufacturing system. The first variant is designed based on Passino's swimming strategy, reproduction strategy and elimination–dispersal strategy, while, the BFA2 uses the modified swimming strategy, reproduction strategy and elimination–dispersal strategy. The computational results reveal that the performance of BFA2 is better than BFA1 and the genetic algorithm.

A step towards solving multi-objective dynamic cellular manufacturing system, Nouri [54] developed a new efficient multi-objective matrix-based BFA with traced constraint handling, known as MOMBATCH. Liu et al. [39] developed an efficient hybrid BFA for solving a combined decision model of a dynamic cellular manufacturing system. The decision model considers the worker assignment as well as production planning for taking the decision. In the proposed algorithm, a two-phase based heuristic is embedded with the proposed algorithm to obtain the initial good quality solution. Liu et al. [47] proposed a hybrid BFA combining a two-phase based heuristic and BFA for a joint decision model of worker assignment and planning of product in a dynamic manufacturing cell. The objective of the problem is to minimize the backorder cost and the holding cost of inventory. In their study, they used elaborately designed solution representation and bacteria evolution operators in tumbling, reproduction and elimination–dispersal to perform the hybrid BFA effectively. They have shown that their algorithm is superior to the original BFA, the discrete BFA, hybrid genetic algorithm and hybrid simulated annealing.

Liu et al. [48] also proposed an integrated BFA embedding a five-phase based heuristic (FPBH) to solve discrete problems of cellular manufacturing system in supply chain. The proposed algorithm incorporates the modified bacteria evolution operators including chemotaxis (swimming), reproduction, and elimination–dispersal and the FPBH. The modified reproduction strategy and the elimination–dispersal strategy keep the healthiest bacterium for subsequent generation and increase the diversity of population. The results of computational experiment demonstrate that the proposed algorithm performs better than the genetic algorithm and the simulated annealing within the same computational time.

Chana [44] designed a BFA based hyper-heuristic algorithm for scheduling of resources in grid computing. In the proposed algorithm, each of the bacteria was represented by a heuristic (i.e. select, move, swap, and drop) or a combination of heuristics. The performance of the proposed algorithm is better than the existing algorithms in the literature. Prakash and Vidyarthi [50] proposed a hybrid BFA by combining genetic algorithm with the BFA to optimize a schedule of jobs in computational grid. The hybrid algorithm enhances the searching capability of the problem involving a large space to converge the optimal or near-optimal solution effectively. Devi and Geethanjali [45] introduced a modified BFA for minimizing total power loss and for improving voltage profile of radial distribution system with distributed generation unit. In their algorithm, they implemented the elimination–dispersal step by computing the difference between the so far obtained best and worst values.

Atasagun and Kara [49] proposed a new discrete BFA for simple straight and U-shaped assembly line balancing problems. In the proposed algorithm, the weights assigned to each task are updated in the chemotaxis strategy and the reproduction strategy by utilizing the basic concept of foraging strategies. The algorithm has been compared with the ant colony optimization-based heuristic and genetic algorithm-based heuristic.

The literature survey reveals the successful application of the BFA in a wide variety of scheduling problems. Therefore, the research is motivated to validate the suitability of the potential application of the BFA algorithm to solve the 2-machine RCC scheduling problem with sequence-dependent setup times. Inspired by this motivation, this paper presents an improved IBFA to solve this scheduling problem. We investigated the comparative performance of the proposed algorithm with some best-known metaheuristic algorithms. We considered the existing algorithms, namely, GA1 [9], SA [10], DCS [11], and two variants of the BFA, basic-BFA [12] and CLS-BFA [12].

3. The proposed algorithm

In this section, we first describe the basic BFA algorithm. Then, we propose an improved IBFA to solve the RCC scheduling problem.

3.1. Basic BFA

BFA developed by Passino [31] is a swarm inspired optimization algorithm based on the group foraging strategy of e-coli bacteria. Recently, it has drawn attention of researchers due to its wide applications in engineering and management field. In real life, each bacterium moves in a direction by a set of tensile flagella. The flagella help the bacteria to perform operations, namely, tumble and swim during foraging. If the bacterium rotates its flagella in clockwise direction, the bacterium tumbles to move slowly in a nutrient gradient. Usually a frequent tumbling helps the bacterium to get away from a harmful place. On the other hand, by moving the flagella in counter clockwise, the bacterium swims to move in a faster rate. When the bacteria find sufficient amount of food, the length of the bacteria increases and at suitable temperature, it splits to form its replica. Thus, by taking inspiration from this phenomenon of e-coli bacteria, Passino [31] considered four major steps such as chemotaxis, swarming, reproduction, and elimination–dispersal in the BFA. Chen et al. [36,55] provided the details of these processes of the classical BFA.

A brief description of the steps of the BFA, namely, chemotaxis, swarming, reproduction, and elimination–dispersal is given as follows:

3.1.1. Chemotaxis

In the BFA, the chemotaxis strategy enables bacteria to move by swimming and tumbling. Suppose $\theta^i(j,k,l)$ represents the position of the ith bacterium at jth chemotactic, kth reproduction and lth elimination–dispersal step. Then, the movement of ith bacterium by swimming and tumbling in (j + 1)th computational chemotaxis can be represented as:

$$\theta^i(j+1,k,l) = \theta^i(j,k,l) + C(i)\frac{\Delta(i)}{\sqrt{\Delta^T(i)\,\Delta(i)}} \tag{3}$$

where, $C(i)$ is the size of the step taken by the i-th bacterium in the random direction specified by the tumble, $\frac{\Delta(i)}{\sqrt{\Delta^T(i)\Delta(i)}}$ indicates the unit vector and Δ(i) indicates a vector in the random direction whose elements lie in [−1, 1].

3.1.2. Swarming

During the movements of the bacteria, they release attractants and repellents to signal other bacteria so that they should swarm together, provided they get nutrient-rich environment or avoided the noxious environment. BFA applies this process by representing the cell-to-cell attraction and repelling effect. The cell-to-cell signaling in *E. coli* swarm is presented as:

$$\begin{aligned} J_{CC}\left(\theta^q(j,k,l),P(j,k,l)\right) &= \sum_{i=1}^{n} J_{CC}\left(\theta^q(j,k,l),\theta^i(j,k,l)\right) \\ &= \sum_{i=1}^{n}\left[-d_{at}e^{\left(-w_{at}\sum_{m=1}^{p}\left(\theta_m^q-\theta_m^i\right)^2\right)}\right] \\ &\quad + \sum_{i=1}^{n}\left[-h_{rp}e^{\left(-w_{rp}\sum_{m=1}^{p}\left(\theta_m^q-\theta_m^i\right)^2\right)}\right] \end{aligned} \tag{4}$$

where, $J_{cc}(\theta^q(j,k,l),P(j,k,l))$ is the time varying fitness function required to add with the actual fitness function (J(q, j, k, l)) in order to represent the release of attractant–repulsion chemicals by the cells during movement. n, p, $\theta^q(j,k,l) = [\theta_1^q(j,k,l), \theta_2^q(j,k,l), \ldots, \theta_p^q(j,k,l)]^T$, θ_m^i and θ_m^q represent total number of bacteria, number of variables in the search space, a point on the optimization domain having p dimension, mth component of ith bacterium position (θ^i) and mth component of qth bacterium position (θ^q) respectively. The parameters d_{at}, w_{at}, h_{rp} and w_{rp} are different coefficients used for signaling, termed as attractant signal depth, attractant signal depth, repellent signal height and repellent signal width respectively.

Compute $J(q,j,k,l)$ and then let $J(q,j,k,l) = J(q,j,k,l) + J_{CC}(\theta^q(j,k,l),P(j,k,l))$.

3.1.3. Reproduction

A fraction of worst bacteria die and new ones are constructed from the remaining healthy bacteria in the population so that the population size remains constant.

3.1.4. Elimination–dispersal

Due to gradual or sudden change in the local environment, the life of the bacteria may be affected. Therefore, in order to incorporate this phenomenon, we eliminate each bacterium in the population with the probability P_{ed} and a new replacement is randomly initialized over the search space.

3.2. The proposed IBFA

The BFA has been successfully applied as a global optimization algorithm to solve a wide range of problems in continuous search domain. Since the RCS problems are usually defined in the discrete search domain, a discrete version of BFA is desirable. In this paper, we propose a discrete IBFA to generate new sequences of parts by converting the continuous positions of the bacteria and the cell-to-cell signals into a discrete permutation sequence of parts. Fig. 1 shows the flowchart of the proposed IBFA algorithm. Before describing the proposed algorithm, we first provide the notations used in the algorithm (see Table 1).

We now describe the proposed algorithm to minimize the cycle time in 2-machine RCC scheduling problems.

1: Initialize the process parameters N, n_c, n_s, n_r, n_{ed}, p_{ed}, w_{at}, d_{at}, h_{rp}, w_{rp}.

Table 1
Nomenclature for the proposed IBFA algorithm.

N	:	Population size
n_c	:	Maximum number of chemotactic steps
n_s	:	Swimming length
n_r	:	Maximum number of reproduction steps
n_{ed}	:	Maximum number of elimination–dispersal steps
p_{ed}	:	Elimination–dispersal probability
w_{at}	:	Attractant signal width
d_{at}	:	Attractant signal depth
h_{rp}	:	Repellent signal height
w_{rp}:	:	Repellent signal width
CT	:	Completion time
CT^i		Completion time of ith bacterium
$J(i,j,k,l)$	:	Fitness function for ith bacterium at jth chemotactic, kth reproduction and lth elimination–dispersal step
P(j, k, l)	:	Positions of all bacteria present in population i.e, [$\theta(1,j,k,l)$, $\theta(2,j,k,l)$,…$\theta(n,j,k,l)$]
$J_{cc}(\theta(i,j,k,l)$,P(j, k, l)):		Time varying fitness function for ith bacterium at jth chemotactic, kth reproduction and lth elimination–dispersal step
J_{best}	:	Best objective function value i.e., $\min\left(CT^1, CT^2, \ldots, CT^n\right)$
LB	:	Lower bound
J_{health} (i, k, l)	:	Health of ith at kth reproduction and lth elimination–dispersal step and can be represent by, $J_{health}(i,k,l) = \sum_{j=1}^{N_c+1} J(i,j,k,l)$

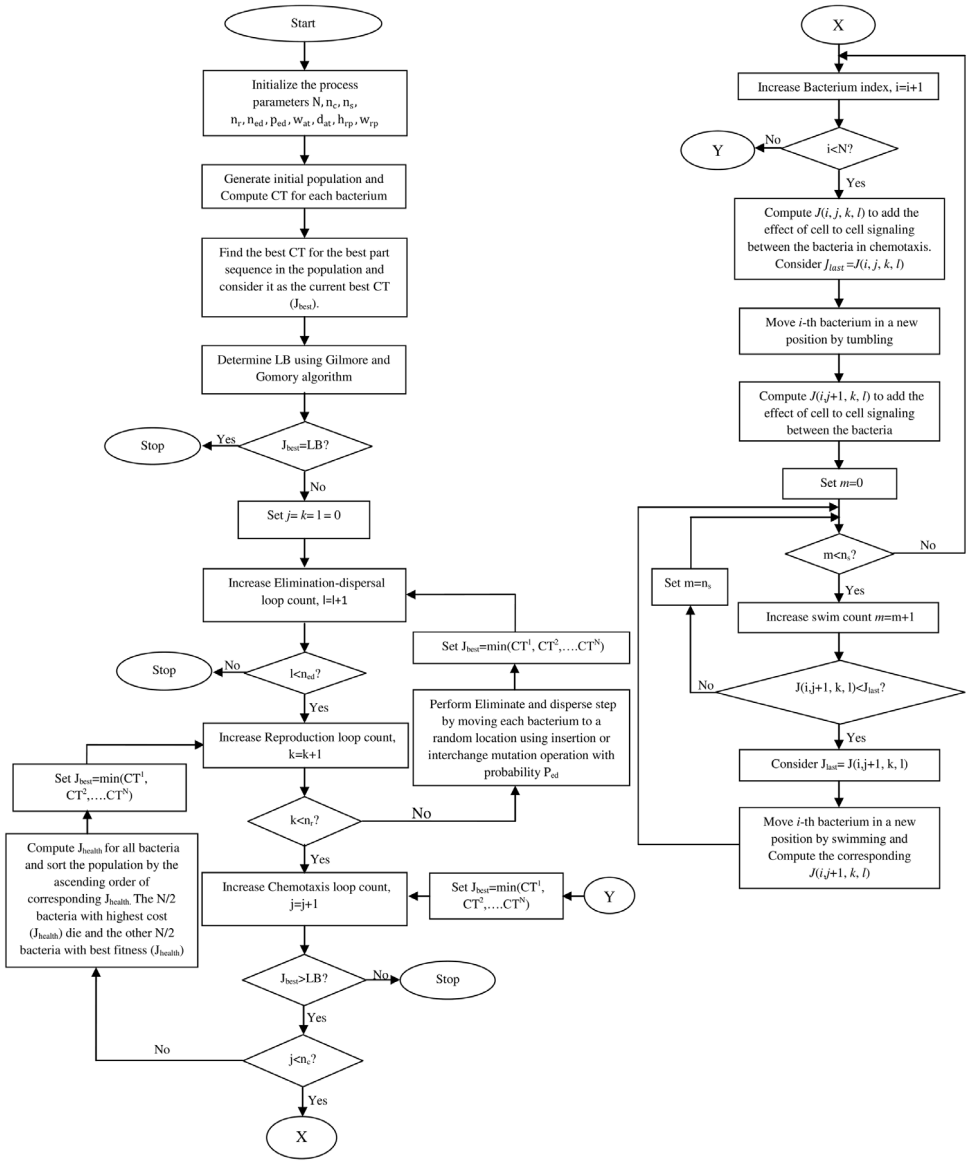

Fig. 1. Flowchart of the proposed IBFA algorithm.

Table 2
An example of calculation of elemental difference between the two sequences.

Initial bacterium Position	Number of different elements between the two sequences	Discrete difference, (θ-θ^t) for Eq. (2)
θ^1: [1-2-4-3-5-7-8-6] θ^2: [1-3-4-2-5-7-8-6]	θ^1: [1-**2**-4-**3**-5-7-8-6] θ^2: [1-**3**-4-**2**-5-7-8-6]	**2**

2: Generate initial population of n part sequences representing N bacteria (described in Section 3.2.1). Compute the CT of each job sequence, which indicates the quality/fitness of each bacterium at that position. Determine the best CT for the best part sequence in the population and consider it as the current best CT (J_{best}).
3: Determine the LB of CT based on Gilmore and Gomory algorithm [8]. If the J_{best} is equal to LB, terminate the procedure and consider the J_{best} as the optimal solution.
4: Set $j = k = l = 0$.
5: Elimination–dispersal loop: $l = l + 1$
6: Reproduction loop: $k = k + 1$
7: Chemotaxis loop: $j = j + 1$
8: For $i = 1, 2, \ldots, N$, apply chemotactic step j for ith bacterium as:

(i) Compute $J(i, j, k, l)$ to add the effect of cell to cell signaling between the bacteria in chemotaxis. Let $J(i, j, k, l) = J(i, j, k, l) + J_{cc}(\theta(i, j, k, l), P(j, k, l))$. Consider $J(i, j, k, l)$ as J_{last}.
(ii) Tumble: Move ith bacterium in a new position by tumbling. Let $\theta(i, j + 1, k, l) = f_1(\theta(i,j, k, l), C(i))$ (described in Section 3.2.2).
(iii) Compute $J(i, j + 1, k, l) = J(i, j + 1, k, l) + J_{cc}(\theta(i, j + 1, k, l),P(j, k, l))$ (given in Section 3.2.3).
(iv) Swim: Steps used to move ith bacterium in a new position by swimming operation are as follows.

- Set $m = 0$.
- While $m < n_s$
- Set $m = m + 1$.

If $J(i, j + 1, k, l) < J_{last}$ then $J_{last} = J_{new}$ $(i, j + 1, k, l)$. Let $\theta(i, j + 1, k, l)$=$f_1(\theta(i, j, k, l),C(i))$ (described in Section 3.2.2). Compute the corresponding $J(i, j + 1, k, l)$. Let $J(i, j, k, l)$ =$J(i, j, k, l) + J_{cc}(\theta(i, j, k, l),P(j, k, l))$ (described in Section 3.2.3).

Table 3
An example of pairwise *interchange mutation and* cyclic shift neighborhood *mutation.*

Position of *i*th bacterium at *j*th chemotactic, *k*th reproductive and *l*th elimination–dispersal step	Pairwise *interchange mutation*	Cyclic shift neighborhood *mutation*
$\theta^i(j,k,l)$: [3-2-1-4-5-7-8-6]	$\theta^i(j,k,l)$: [3-**2**-1-**4**-5-7-8-6] $\theta^i(j+1,k,l)$: [3-**4**-1-**2**-5-7-8-6]	$\theta^i(j,k,l)$: [3-**2**-**1**-4-5-7-8-6] $\theta^i(j,k,l+1)$: [3-**1**-**2**-4-5-7-8-6]

Table 4
Tuning process parameters and their levels of the proposed IBFA method.

Discrete swimming and tumbling during chemotaxis — discrete random walk during elimination–dispersal	$w_{attract}$ (A)	$w_{repellent}$ (B)	P_{ed} (C)	% of Closeness (D)
Level(shift–shift): cyclic shift neighborhood-cyclic shift neighborhood	Level (−2):8.5	Level (−2):10	Level (−2):0.95	Level (−2):0.981
Level(Shift–interchange):cyclic shift neighborhood-pairwise *interchange*	Level (−1):6.425	Level (−1):8.75	Level(−1):0.775	Level (−1):0.94075
Level(interchange–shift):pairwise *interchange*-cyclic shift neighborhood	Level (0):4.36	Level (0):7.5	Level (0):0.6	Level (0):0.9005
Level(interchange–interchange): pairwise *interchange*-pairwise *interchange*	Level (1):2.275 Level (−2):0.2	Level (1):6.25 Level (−2):5	Level (1):0.425 Level (−2):0.25	Level (1):0.86025 Level (−2):0.82

Table 5
ANOVA of the generated near-optimal solution.

Source	DF	Adj SS	Adj MS	F-Value	P-Value
Model	29	13123799	452545	410.93	0.000
Error	94	103520	1101	1.33	0.223
Lack-of-Fit	70	82251	1175		
Pure Error	24	21269	886		
Total	123	13227320			
R^2		99.22%	R^2adj		98.98%

Else, Let $m = n_s$. (End of while loop for swimming operation)

(v) Go to next $(i+1)$-bacterium if $i \neq N$ and set $i = i+1$.

9: Set $J_{best} = \min\left(CT^1, CT^2, \ldots, CT^N\right)$.
10: If $J_{best} >$ LB, go to step 11, else go to step 17.
11: If $j < n_c$, return to step 7.
12: Reproduction (given in Section 3.2.4):

(i) For $i = 1, 2,\ldots, n$, calculate $J_{health}(i,k,l)$. We consider $J_{health}(i,k,l) = \sum_{j=1}^{N_c+1} J(i,j,k,l)$. This denotes the number of nutrient places visited by each bacterium during chemotaxis cycle. Sort the population of bacteria and chemotactic parameters $C(i)$ by the ascending order of corresponding J_{health}.

(ii) The $N/2$ bacteria with highest cost (J_{health}) die and the other $N/2$ bacteria with best fitness (J_{health}) values split.

(iii) Set $J_{best} = \min\left(CT^1, CT^2, \ldots, CT^N\right)$.

13: If $k < n_r$, return to step 6.
14: Elimination and dispersal: Eliminate and disperse *i*th bacterium to random location using insertion or interchange mutation operation (described in Section 3.2.5) with probability p_{ed}.
15: Set $J_{best} = \min\left(CT^1, CT^2, \ldots, CT^N\right)$.
16: If $l < n_{ed}$, return to step 5.
17: Post process and obtain the optimal part sequence for minimum *CT*.

3.2.1. Representation of bacteria and initial population

In the proposed algorithm, we consider a permutation sequence of parts in the RCS problem as the position of each bacterium, $\theta^i(j,k,l)$. The initial population comprises N distinct randomly generated part sequences. These N sequences represent N bacteria in the population. The cycle time of each sequence indicates the quality/fitness of each bacterium.

3.2.2. Movement of bacteria in computational chemotaxis

In the traditional BFA, the bacterium moves for swimming and tumbling in a continuous search space following Eq. (3). During the chemotaxis strategy, the position of each bacterium is updated by considering Eq. (3) shown in Section 3.1.1. However, from Eq. (3), it has been observed that the updated position of each bacterium may have fractional numeric values, which clearly indicates that the search process is carried out in a continuous search space. In addition, the positive and negative values of Δ in Eq. (3) indicate that the direction of movement of each bacterium is in continuous search space. Since in the present study, the position of each bacterium is represented by a part sequence and the bacterium moves in discrete search space, we do not consider Eq. (3) for the chemotaxis. Therefore, in the proposed algorithm, instead of using Eq. (3), a modified equation given by (5) is used. In this equation, f1 represents the mutation operation using two neighborhood search techniques such as pairwise interchange and cyclical shift methods and C represents the step size, i.e., the number of random pairwise interchange or cyclic shift neighborhood operations during each mutation operation. Thus, the modified Eq. (5) enables the algorithm to search in a discrete search space. Moreover, the fractional value of C obtained from Eq. (6) is converted to integer value, which helps the bacterium to search in a discrete search space. The position of the bacterium by swimming and tumbling in computational chemotaxis is given as

$$\theta^i(j+1,k,l) = f_1(C, \theta^i(j,k,l)) \tag{5}$$

where, f_1 represents the mutation operation using two neighborhood search techniques, pairwise interchange and cyclical shift methods. Step size, *C* represents the number of random pairwise interchange or cyclic shift neighborhood during each mutation operation. $\theta^i(j,k,l)$ is the initial job sequence as bacterial position.

The pairwise interchange neighborhood swaps randomly any two jobs in the sequence. For example given in Table 3, the initial job sequence ($\theta^i(j,k,l)$) is [3-**2**-1-**4**-5-7-8-6]. Suppose, C (number of pairwise interchange) = 1. Now, applying the pairwise interchange neighborhood, we select two random positions, say, 2 and 4 and the corresponding jobs 2 and 4 in the initial job sequence are interchanged, resulting in a new job sequence $\theta^i(j+1,k,l)$ as [3-**4**-1-**2**-5-7-8-6]. On the other hand, as given in Table 3, we consider a job sequence ($\theta^i(j,k,l)$) as [3-**2**-**1**-4-5-7-8-6] and apply the cyclic shift neighborhood. Suppose, C (number of pairwise interchange) = 1. We first select one random position in the $\theta^i(j,k,l)$, say, 2. Now, select the job in position 2 and the job corresponding to its adjacent position and interchange with each other. Therefore, the generated job sequence ($\theta^i(j+1,k,l)$) as [3-**1**-**2**-4-5-7-8-6] is obtained.

Sometimes, when the bacterium cell is near the optimum, it suffers from sustained oscillations, especially for flat fitness landscape. To accelerate the convergence speed near optima, the

adaptive step size C [56] is considered in this study. The adaptive step size C for the chemotactic is given as follows

$$C = \frac{1}{1 + \frac{\lambda}{|J(\theta) - J_{best}|}} \tag{6}$$

where, the λ value depends on the LB and % of Closeness of the solution of the problem and is obtained from the following expression.

$$\lambda = 1 + \left[\frac{LB}{\%of\ Closeness} - LB\right] \tag{7}$$

where, $\left[\frac{LB}{\%of\ Closeness} - LB\right]$ represents the approximate gap between the exact optimal solution and the LB.

In this case, the LB of the CT is used to verify the performance of an algorithm, particularly for large-sized problems with no available optimal solutions produced by the exact methods and it is computed using Gilmore and Gomory algorithm [8]. The idea for computing the LB is that the problem with sequence-dependent setup times should be relaxed to a sequence independent one. The procedural steps for computing the LB (given in [8] are as follows:

Step 1: For each job, j in MPS set, loading time on machine 1, $\alpha_j^{11} = \min_{1\le i\le n} \alpha_{ij}^{11}$ and loading time on machine 2, $\alpha_j^{21} = \min_{1\le i\le n} \alpha_{ij}^{21}$

Step 2: Find: $\gamma = 6n\beta + \sum_{j=1}^{n}\left(\alpha_j^{Input} + \alpha_j^{11} + \alpha_j^{12} + \alpha_j^{21} + \alpha_j^{22} + \alpha_j^{Output}\right)$

Where,

n = Total number of jobs in MPS

β = Time required for robot to travel from Input buffer to Machine 1, Machine 1 to Machine 2 and Machine 2 to Output buffers

α_j^{Input} = Unloading time of jth job from Input buffer

α_j^{12} = Unloading time of jth job from Machine 1

α_j^{22} = Unloading time of jth job from Machine 2

α_j^{Output} = Unloading time of jth job from Output buffer

Step 3: Find: $\vartheta = \min_{1\le j\le n} P_j^1 + \min_{1\le j\le n} P_j^2$

Where,

P_j^1 = Processing time of jth job in Machine 1

P_j^2 = Processing time of jth job in Machine 2

Step 4: Determine ψ_1 and ψ_2 using equations given below:

$$\psi_1 = \min_{1\le j\le n} P_j^1 - 3\beta - \max_{1\le j\le n} \alpha_j^{Input} - \max_{1\le j\le n} \alpha_j^{11}$$

$$\psi_2 = \min_{1\le j\le n} P_j^2 - 3\beta - \max_{1\le j\le n} \alpha_j^{22} - \max_{1\le j\le n} \alpha_j^{Output}$$

Step 5: For all jobs in MPS find g_j and h_j from equation shown below:

$$g_j = \max\{\vartheta, P_j^1 + \psi_2\}$$

$$h_j = P_j^2 + \psi_1$$

Step 6: Compute, CT^*_{GnG}: $\min\{CT_{GnG} = \sum_{l=1}^{n}(g_{l+1}, h_l)\}$ using Gilmore and Gomory (GnG) algorithm.

Step 7: Determine the Lower Bound: $LB = \gamma + CT^*_{GnG}$

The LB is equal or less then the optimum solution of a problem. The percentage of closeness is defined as:

$$\text{closeness}(\%) = \frac{LB}{\text{Optimal solution}} \times 100 \tag{8}$$

In the present study, since the optimal solution is not available for large job sized problems, it is not possible to determine the value of closeness (%) for these jobs. If the gap between the optimal solution and the LB is equal to zero, then $\lambda = 1$. Moreover, the LB is used in place of J_{best}. Thus, the modified step size C in the present study can be obtained as:

$$C = \begin{cases} 1 & \text{if } \frac{1}{1+\frac{1+\left[\frac{LB}{\%of\ Closeness} - LB\right]}{|J(\theta) - LB|}} \ge 0.5 \\ 0 & \text{if } \frac{1}{1+\frac{1+\left[\frac{LB}{\%of\ Closeness} - LB\right]}{|J(\theta) - LB|}} < 0.5 \end{cases} \tag{9}$$

If a bacterium is far away from the global best, then $|J(\theta) - LB| > 1 + \left[\frac{LB}{\%of\ Closeness} - LB\right]$, making $\frac{1}{1+\frac{1+\left[\frac{LB}{\%of\ Closeness} - LB\right]}{|J(\theta) - LB|}} \ge 0.5$ and $C = 1$. Similarly, if a bacterium is very close to optima, then $|J(\theta) - LB| < 1 + \left[\frac{LB}{\%of\ Closeness} - LB\right]$ makes the denominator very large, resulting in $C = 0$.

It is true that for most of the problem instances taken under consideration, the value of C becomes one. This is due to small possibility to get the optimal or near optimal solution for large-sized problems. However, during convergence, if the best bacterium reaches the optimum or near optimum solution, then, according to Eq. (9), the value of C will be equal to zero and the movement of the bacterium stops. Fig. 5 illustrates the convergence of the solution and simultaneous change in C value with respect to number of iterations. This reduces the unnecessary oscillatory movement of bacterium and thus, the convergence speed increases. However, if C becomes zero at near optimum region, the elimination–dispersal strategy helps the bacterium to reach optimum.

The C value shown in Fig. 5 indicates the change in C value of the best bacterium with respect to the number of iterations. The figure shows that at the initial stage, as the best bacterium is far away from the optimum position, the C value of the best bacterium becomes one. However, when the best bacterium reaches to near-optimum solution (here, 4633), it brings to a standstill position during the chemotaxis step. This reduces the unnecessary oscillatory movement of the best bacterium near the optimal position and subsequently, increases the computational speed. However, due to this standstill behavior, the best bacterium may get stuck in a local optimum zone. To get rid of the local optimum, the bacterium moves in the elimination–dispersal step.

3.2.3. Swarming (computation of J_{cc})

During cell-to-cell communication, the difference between the positions of two bacteria is taken as the total number of elemental differences between the two sequences. To illustrate this procedure, we consider a sequence of 8 parts, [2-3-6-8-5-7-4-1] as the position (θ^1) of a bacterium (B_1). Similarly, the positions of two other bacteria (B_2 and B_3) are taken as: $\theta 2$ = [1-3-2-8-5-7-4-6] and θ^3 = [2-3-6-5-8-7-4-1]. Now, to obtain the value of J_{CC} of B_1 with respect to B_2 and B_3, we calculate the number of different parts between the positions of θ^1 with θ^2 and θ^1 with θ^3. The number of different parts thus obtained for $(\theta^1 - \theta^2) = 3$ and for $(\theta^1 - \theta^3) = 2$. Finally, the value of J_{cc} for B_1 is calculated using Eq. (4) as $J_{cc} = 0.0033$, where, $d_{at} = h_{rp} = 0.5$, $w_{at} = 8.5$, and $w_{rp} = 5$. Another example to demonstrate the calculation of number of different parts between two sequences θ^1 and θ^2 is given in Table 2. The different parts in the sequences are shown bold.

3.2.4. Reproduction

The population (N) is sorted in ascending order of accumulated fitness value of the bacteria. Then, after removing $N/2$ least healthy bacteria from the population, the other $N/2$ healthiest bacteria is considered for reproduction and after splitting each of them into two bacteria, they are placed at the same location, keeping the population of the bacteria constant.

Table 6
ANOVA of convergence time.

Source	DF	Adj SS	Adj MS	F-Value	P-Value
Model	29	32.1092	1.1072	72.52	0.000
Error	94	1.4352	0.0153		
Lack-of-Fit	70	1.3240	0.0189	4.08	0.000
Pure Error	24	0.1113	0.0046		
Total	123	33.5444			
R^2		95.72%		R^2adj	94.40%

3.2.5. Elimination–dispersal

Similarly, during the elimination-dispersal operation, the pair-wise interchange or cyclic shift neighborhood mutation is used for random movement of each bacterium. To illustrate the movement of the bacteria during the chemotaxis and elimination-dispersal steps, an example is shown in Table 3.

4. Tuning the parameters of the IBFA

Since the performance of the proposed IBFA greatly depends on its process parameters, an effort was made to choose the optimum setting of its process parameters. The process parameters considered in the IBFA include mutation used for discrete swimming/ tumbling and discrete random walk during the elimination–dispersal step, the parameters, namely, w_{at}, w_{rp}, p_{ed} and % of Closeness. Among these parameters, w_{at}, w_{rp}, p_{ed} and % of Closeness were taken as the continuous factor, whereas, mutation operations used for discrete swimming/tumbling and discrete random walk during the elimination–dispersal step are considered as categorical factors. The performance parameters of the IBFA are taken as solution quality and CPU time. A desirability function-based response surface methodology is applied to optimize both performance parameters simultaneously and to enhance the convergence speed of the proposed algorithm. In addition, a design of experiment based on central composite design is employed to produce dataset for tuning the operating parameters of the proposed algorithm having cube point = 32, center points in cube = 12 and axial point = 20. The tuning process parameters and their levels of the proposed algorithm is shown in Table 4.

A regression analysis is chosen to establish correlation between the factors and the response parameters from the experimental dataset and it was performed on Minitab 17. The second order regression models thus obtained are given as follows:

Discrete swimming and tumbling using cyclic shift neighborhood and discrete random walk using cyclic shift neighborhood

$$\begin{aligned}\text{Solution} = {} & 11913 + 8.5x_1 + 76.1x_2 + 476x_3 + 10576x_4 \\ & - 0.713x_1^2 - 0.76x_2^2 - 14x_3^2 - 5463x_4^2 - 2.23x_1x_2 \\ & + 14.8x_1x_3 + 18.1x_1x_4 + 12.2x_2x_3 - 67.4x_2x_4 \\ & - 650x_3x_4 \end{aligned} \quad (10)$$

$$\begin{aligned}\text{CPU time} = {} & 20.33 - 0.061x_1 + 0.276x_2 + 2.63x_3 - 9.5x_4 \\ & - 0.00283x_1^2 + 0.00095\,x_2^2 - 0.526x_3^2 + 6.94x_4^2 \\ & - 0.00305x_1x_2 + 0.0436x_1x_3 + +0.098x_1x_4 + \\ & - 0.0522x_2x_3 - 0.281x_2x_4 - 1.97x_3x_4 \end{aligned} \quad (11)$$

Discrete swimming and tumbling using cyclic shift neighborhood and discrete random walk using pair-wise *interchange* neighborhood:

$$\begin{aligned}\text{Solution} = {} & 11521 + 2.7x_1 + 72.0x_2 + 423x_3 + 11074x_4 \\ & - 0.713x_1^2 - 0.76x_2^2 - 14x_3^2 - 5463x_4^2 - 2.23x_1x_2 \\ & + 14.8x_1x_3 + 18.1x_1x_4 + 12.2x_2x_3 - 67.4x_2x_4 \\ & - 650x_3x_4 \end{aligned} \quad (12)$$

$$\begin{aligned}\text{CPU time} = {} & 19.44 - 0.072x_1 + 0.274x_2 + 2.69x_3 - 8.4x_4 \\ & - 0.00283x_1^2 + 0.00095x_2^2 - 0.526x_3^2 + 6.94x_4^2 \\ & - 0.00305x_1x_2 + 0.0436x_1x_3 + 0.098x_1x_4 \\ & - 0.0522x_2x_3 - 0.281x_2x_4 - 1.97x_3x_4 \end{aligned} \quad (13)$$

Discrete swimming and tumbling using pair-wise *interchange* and discrete random walk using cyclic shift neighborhood:

$$\begin{aligned}\text{Solution} = {} & 11283 - 1.3x_1 + 78.9x_2 + 440x_3 + 10583x_4 \\ & - 0.713x_1^2 - 0.76x_2^2 - 14x_3^2 - 5463x_4^2 - 2.23x_1x_2 \\ & + 14.8x_1x_3 + 18.1x_1x_4 + 12.2x_2x_3 - 67.4x_2x_4 \\ & - 650x_3x_4 \end{aligned} \quad (14)$$

$$\begin{aligned}\text{CPU time} = {} & 19.66 - 0.073x_1 + 0.277x_2 + 2.65x_3 - 9.8\times x_4 \\ & - 0.00283x_1^2 + 0.00095x_2^2 - 0.526x_3^2 + 6.94x_4^2 \\ & - 0.00305x_1x_2 + 0.0436x_1x_3 + 0.098x_1x_4 \\ & - 0.0522x_2x_3 - 0.281x_2x_4 - 1.97x_3x_4 \end{aligned} \quad (15)$$

Discrete swimming and tumbling using pair-wise *interchange* and discrete random walk using pair-wise *interchange neighborhood:*

$$\begin{aligned}\text{Solution} = {} & 11188 - 1.3x_1 + 78.9x_2 + 429x_3 + 10701x_4 \\ & - 0.713x_1^2 - 0.76x_2^2 - 14x_3^2 - 5463x_4^2 - 2.23x_1x_2 \\ & + 14.8x_1x_3 + 18.1x_1x_4 + 12.2x_2x_3 - 67.4x_2x_4 \\ & - 650x_3x_4 \end{aligned} \quad (16)$$

$$\begin{aligned}\text{CPU time} = {} & 20.24 - 0.097x_1 + 0.246x_2 + 2.54x_3 - 10.1x_4 \\ & - 0.00283x_1^2 + 0.00095x_2^2 - 0.526x_3^2 + 6.94x_4^2 \\ & - 0.00305x_1x_2 + 0.0436x_1x_3 + 0.098x_1x_4 \\ & - 0.0522x_2x_3 - 0.281x_2x_4 - 1.97x_3x_4 \end{aligned} \quad (17)$$

The statistical accuracy of the mathematical models developed is tested using ANOVA analysis at 95% confidence level. From the results of the analysis given in Tables 5 and 6, it is observed that the calculated p-value for both the models is less than 0.05, which indicates the adequacy of the models. In addition, the results of the coefficients of determination R^2 and adjusted R^2 of the models show that the regression models are highly significant as well.

The mathematical models thus obtained are then used with desirability function based response surface method for optimizing both solution and CPU time simultaneously. The three sets of equations for minimization of the response, hitting a target value between lower and upper limit, and maximization of the response used to calculate individual desirability index applied in this experimentation are obtained from Majumder and Laha [11]. Since the objective of the study is to minimize both the response parameters, the individual desirability index was selected based on the minimum value of the objective function. On the other hand, to find the non-dominated solution characteristic and to choose the lower and upper bound of the response parameters by giving more importance to solution parameter, an approximate Pareto front was constructed using NSGA-II [57] as given in Fig. 2. It is seen that in between the region between A and B, the optimal value of solution dominates over the optimal value of CPU time. Therefore, the upper bound of the solution and the lower bound of CPU time are chosen from the end point (point B) of the region. It reveals from Fig. 2 that 16314 is selected as upper bound of the solution and 16.64 is taken as the lower bound of CPU time.

The generated individual desirability is then combined to determine composite desirability. The equation used for determining the composite desirability is as follows:

$$D = \left(\prod_{i=1}^{n} w_i d_i\right)^{1/\sum_{i=1}^{n} w_i} \quad (18)$$

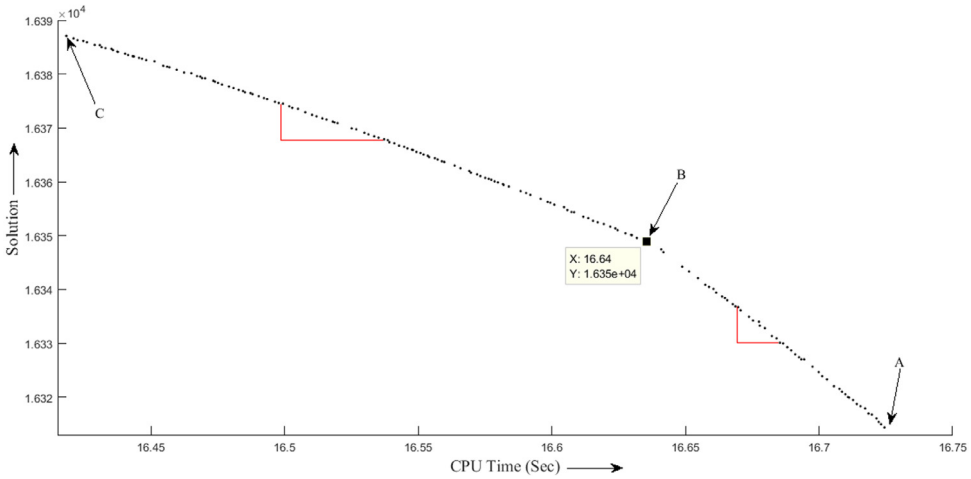

Fig. 2. Approximate Pareto front for the solution and the CPU time.

where, individual weight for each response parameter was taken as 0.5. Thus, the optimal process parameters obtained by the desirability based response surface method of the IBFA (as given in Fig. 2) are as follows: Discrete swimming/tumbling and discrete random walk during elimination–dispersal = Swap-Shift, $w_{at} = 0.2$, $w_{rp} = 5.0505$, $p_{ed} = 0.95$, and % of Closeness = 0.981. While the other parameters: d_{at}, h_{rp}, N, n_c, n_s, n_r and n_{ed} were taken as 0.1, 0.1, 20, 2, 10, 100, and 5 respectively from the paper of Nouri and Hong [38]. The desirability plot showing optimal process parameters of the IBFA is shown in Fig. 4.

5. Non-parametric performance measure

We applied a non-parametric performance measure, namely, two-tailed Wilcoxon Signed Rank test to find the significance of improvement of results produced by the proposed algorithm over the compared algorithms for the Datasets 1, 2 and 3. This technique is used to compare the performance of any two algorithms and the improvement in performance of one algorithm with respect to the other is significant if the obtained p-value is less than 0.05 at 95% confidence level. The results obtained from this test are reported in Tables 9–11. From the results of the tables, it has been seen that the p-values obtained by comparing the performance of the proposed algorithm with each of the existing algorithms varies between 0.00222 to 0.00148, which are considerably less than 0.05. This clearly indicates that the proposed algorithm is statistically significantly better than the existing algorithms.

6. Results and discussion

We evaluated and compared the performance of the proposed IBFA algorithm with some noteworthy metaheuristic algorithms. These comparing algorithms include GA1 [9], SA [10], DCS [11] and two variants of the BFA, basic-BFA [12] and CLS-BFA [12]. We considered these algorithms from the recent literature to compare with the proposed algorithm because these algorithms in the literature are relatively more effective compared to the existing algorithms to solve large-sized test problems in RCS.

Each of these algorithms was coded in MATLAB 2009a programming environment and executed on a PC with Intel i5-2450M CPU with 4 GB RAM running at 2.50 GHz. Table 7 shows the process parameters and stopping criteria considered for the compared algorithms.

Following the same experimental framework as given in Majumder and Laha [11], we generated the test problems randomly where the processing time for a given problem instance follows a discrete uniform probability distribution. These test problem instances were classified into three datasets based on processing times, loading/unloading times and the traveling time (δ) of the

Table 7
Process parameter values and stopping criterion of the compared algorithms.

Algorithm	Process Parameters	Stopping criterion
GA1 (2010)	Population size (N) = 300 Crossover probability = 0.9 Mutation probability = 0.4	Number of generations = 1000 or the maximum number of non-improving generations = 100
SA (2016)	Initial temperature = 665, Dec_Ratio = 0.99 Number of iterations at each temperature = number of parts considered for each instance	Temperature = 0.15
DCS (2016)	N = 20, $P_k = 0.01$, $P_S = 0.6768$, $\lambda = 3$, $P_a = 0.4242$, discrete random walk based on interchange neighborhood operation	Number of generations = 10000 or best generated solution = lower bound
Basic-BFA (2016)	$p_{ed} = 0.25$, $d_{at} = 0.1$, $w_{at} = 0.2$, $h_{rp} = 0.1$, $w_{rp} = 10$	$n_c = 2$, $n_s = 10$, $n_r = 100$, $n_{ed} = 4$
CLS-BFA (2016)	$p_{ed} = 0.25$, $d_{at} = 0.1$, $w_{at} = 0.2$, $h_{rp} = 0.1$, $w_{rp} = 10$	$n_c = 2$, $n_s = 10$, $n_r = 100$, $n_{ed} = 4$

Table 8
Details of experimental framework pertaining to test problems.

Datasets	Processing times	Loading/unloading times	δ
1	[1, 30]	[1,20]	15
2	[1, 100]	[1,30]	50
3	[5, 200]	[5,50]	15

Table 9
Wilcoxon signed rank test for Dataset 1.

Difference between algorithms	W^+	W^-	P-value
IBFA-GA1	0	91	0.00148
IBFA-SA	0	91	0.00148
IBFA-DCS	0	78	0.00222
IBFA-Basic BFA	0	91	0.00148
IBFA-CLS BFA	0	91	0.00148

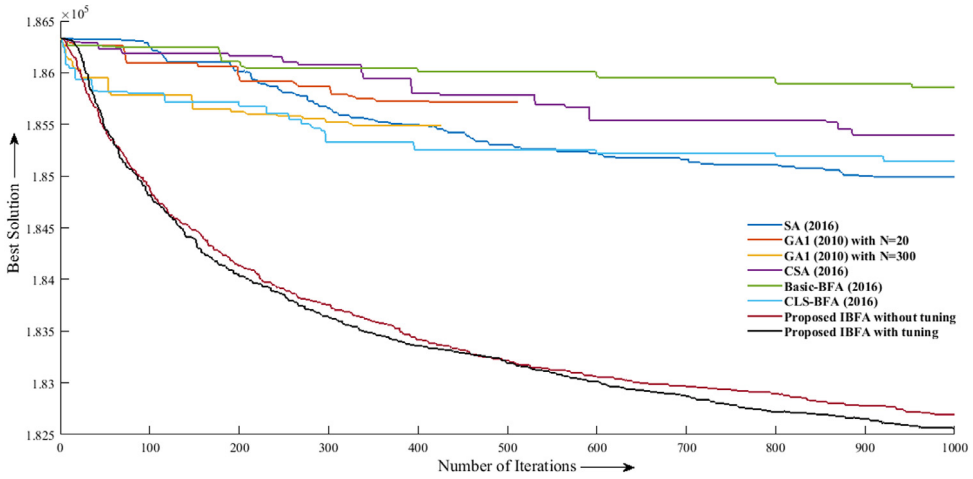

Fig. 3. Convergence characteristics of all the algorithms for the problem of Dataset 2 with part size of 350.

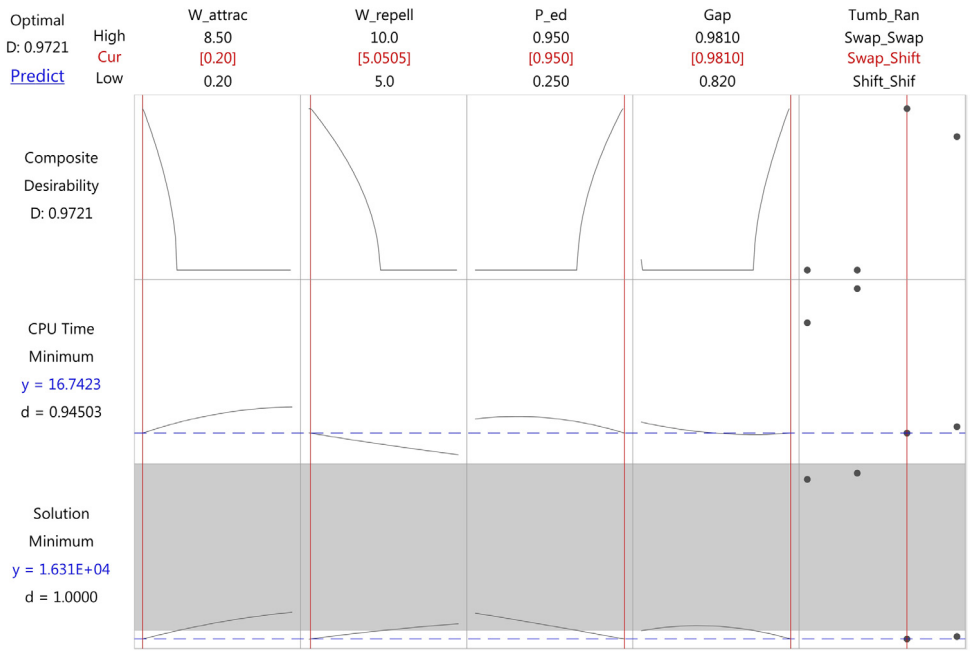

Fig. 4. Desirability plot showing optimal process parameters of the IBFA.

Table 10
Wilcoxon signed rank test for Dataset 2.

Difference between algorithms	W^+	W^-	*P*-value
IBFA-GA1	0	91	0.00148
IBFA-SA	0	91	0.00148
IBFA-DCS	0	91	0.00148
IBFA-Basic BFA	0	91	0.00148
IBFA-CLS BFA	0	91	0.00148

Table 11
Wilcoxon signed rank test for Dataset 3.

Difference between algorithms	W^+	W^-	*P*-value
IBFA-GA1	0	91	0.00148
IBFA-SA	0	91	0.00148
IBFA-DCS	0	91	0.00148
IBFA-Basic BFA	0	91	0.00148
IBFA-CLS BFA	0	91	0.00148

robot between the input hopper and m_1, m_1 and m_2, m_2 and the output hopper. Table 8 provides the details of the information pertaining to these datasets. We considered the number of parts for the problem instances of each these datasets as 200, 225, 250, 275, 300, 325, 350, 375, 400, 425, 450, 475, and 500.

To investigate the relative performance of the proposed method with the compared methods, we considered various situations that deal with the variability of the processing times, loading/unloading times and δ from low to high as given in Table 8. In order to compare the performance of the proposed IBFA with the compared algorithms, we considered two performance measures, namely, absolute value of *CT* given in Eqs. (1)–(2) and the percentage of deviation (*PD*) with respect to the lower bound (*LB*) for any problem instance. The *LB* of *CT* for a given problem instance is computed based on Gilmore and Gomory algorithm and is used to verify the performance of an algorithm on that problem instance. The details of computation of the *LB* are given in Section 3.2.2. Clearly, the best performance of an algorithm is achieved when the *PD* for that algorithm is zero. For a particular

Table 12
Results of the best, worst and mean *CT* values of various algorithms for Dataset 2.

No. of Parts	Basic-BFA (2016)			BFA (after using swapping and shifting mutation)			Proposed IBFA (without tuning)			Proposed IBFA (after tuning)		
	Best	Worst	Mean	Best	Worst	Mean	Best	Worst	Mean	Best	Worst	Mean
200	93670	93900	93741	90309	90396	90350	90455	90502	90484	90311	90450	90371
225	106120	106360	106264	102526	102628	102572	102598	102643	102627	102476	102608	102576
250	116912	117125	117039	112850	112895	112865	101610	101762	101686	100609	100705	100673
275	129432	129700	129549	124945	125040	124990	125103	125207	125162	124931	125120	124997
300	141888	142130	141987	137061	137176	137107	141685	141791	141752	136988	137183	137065
325	152914	153161	153025	147409	147663	147550	147583	147721	147661	147441	147580	147532
350	185856	185987	185934	182588	182649	182615	182631	182692	182652	182562	182623	182592
375	188093	188387	188310	183535	183651	183587	183627	183844	183689	183545	183627	183601
400	199453	199828	199693	194410	194745	194566	194559	194647	194597	194122	194499	194418
425	212899	213103	213026	207729	207863	207773	207801	207954	207853	207682	207755	207731
450	225320	225414	225368	219649	219859	219742	219843	219982	219902	219553	219845	219713
475	237591	237886	237730	231814	232058	231930	232012	232096	232056	231544	231910	231735
500	250501	250590	250549	244468	244587	244563	244578	244707	244633	244190	244518	244402
Average	172358	172582	172478	167638	167785	167708	167237	167350	167289	166612	166802	166724

Table 13
Results of the percentage of deviation of various algorithms for Dataset 2.

No. of Part	Lower Bound	Basic-BFA (2016)		Proposed IBFA (after using swapping and shifting)		Proposed IBFA (without using NSGAII)		Proposed IBFA (using NSGAII)	
		Best *CT*	*PD*	Best *CT*	*PD*	Best *CT*	*PD*	Best *CT*	*PD*
200	73257	93670	27.86	90309	23.28	90455	23.47	90311	23.28
225	82472	106120	28.67	102526	24.32	102598	24.40	102476	24.26
250	91343	116912	27.99	112850	23.54	101610	11.24	100609	10.14
275	100442	129432	28.86	124945	24.39	125103	24.55	124931	24.38
300	109775	141888	29.25	137061	24.86	141685	29.07	136988	24.79
325	119005	152914	28.49	147409	23.87	147583	24.01	147441	23.89
350	154975	185856	19.93	182588	17.82	182631	17.84	182562	17.80
375	150729	188093	24.79	183535	21.76	183627	21.82	183545	21.77
400	160554	199453	24.23	194410	21.09	194559	21.18	194122	20.91
425	170946	212899	24.54	207729	21.52	207801	21.56	207682	21.49
450	181320	225320	24.27	219649	21.14	219843	21.24	219553	21.09
475	191424	237591	24.12	231814	21.10	232012	21.20	231544	20.96
500	200921	250501	24.68	244468	21.67	244578	21.73	244190	21.54
Average	137474	172358	25.97	167638	22.34	167237	21.79	166611	21.25

problem instance, we define the percentage of gap as follows:

$$PD = \left(\frac{CT \text{ generated by the algorithm} - LB}{LB} \right) \times 100 \tag{19}$$

Since all the compared algorithms and the proposed method are non-deterministic metaheuristic algorithms, it would not be reasonable to draw conclusion from the results of single execution each of these methods for a given problem instance. Therefore, the solution reported in this paper for any problem instance was taken as the best, worst and mean of five independent runs of each method.

We adopted some modifications in the proposed IBFA to enhance the results of the basic BFA. The first modification in the proposed algorithm is the mutation operation in the chemotaxis step to allow different motions of the bacterium by exploring more search space and as a result, this operation helps the bacterium to move towards the optimal position. Tables 12 and 13 display the corresponding results. The second modification implemented in the proposed algorithm is the adaptive chemotaxis step size *C*. This *C* value helps the bacteria to avoid unnecessary oscillatory movement after reaching at optimal position. However, the major drawback to consider the adaptive *C* is that the bacteria stop to move in the chemotaxis step when it reaches the local optimum zone. Therefore, at the local optimum, the movement of the bacteria towards optimum position is completely dependent on the mutation operation carried out during the elimination–dispersal step. This reduces the exploration capability and thus, slows down the convergence rate and there is a tendency of being stuck into a local optimum. Tables 12 and 13 show the enhancement of results due to this modification in the proposed algorithm.

We implemented a multi-objective optimization algorithm based on RSM-NSGA-II-Desirability technique to tune the process parameters of the proposed IBFA. The process parameters of the NSGA-II implemented in this study are set: population size = 200, crossover rate = 0.70, mutation rate = 0.02 and number of generations = 200. The details of the NSGA-II are given in Delgarm et al. [58] and Fallah and Honarparast [59]. As a result, the probability of movement of each bacterium increases through the mutation operation during the elimination–dispersal step and consequently, the proposed algorithm performs either better than or comparable with other BFAs in terms of the solution quality. Additionally, a reduction in the computational time is observed from the results of Table 14 when compared with the IBFA without tuning. In the proposed tuned IBFA, the NSGA-II is used to select the upper bound and the lower bound of the responses for the desirability technique. While selecting the upper bound and lower bound of the responses, more weightage is given to the solution quality than the CPU time.

The process parameters of the IBFA before tuning are taken as: $d_{at} = hrp = 0.1$, $w_{at} = 0.1$, $w_{rp} = 10$, number of bacteria (N) = 20, number of chemotaxis steps (n_c) = 2, number of swarming steps (n_s) = 10, number of reproduction steps (n_{re}) = 100, number of elimination–dispersal steps (n_{ed}) = 4, probability of elimination–dispersal (p_{ed}) = 0.5 and % of Closeness = 0.981). The results in Tables 12 and 13 show that before tuning of the IBFA, the solutions generated are comparable to those obtained by the BFA (after using swapping and shifting mutation operations).

Table 14
Average CPU time (in seconds) taken by various algorithms for Dataset 2.

No. of Parts	Basic-BFA (2016)	Proposed IBFA (after using swapping and shifting)	Proposed IBFA (without using NSGAII)	Proposed IBFA (using NSGAII)
200	31.44	58.90	39.23	41.50
225	33.69	64.98	45.67	48.63
250	40.75	71.63	48.90	50.87
275	40.85	77.48	51.98	53.47
300	42.03	84.58	54.50	56.21
325	50.52	88.44	57.42	60.52
350	58.33	94.68	64.20	65.70
375	60.20	102.58	66.43	70.21
400	61.92	104.96	73.96	76.14
425	63.65	116.86	80.27	83.62
450	65.95	126.24	85.12	86.94
475	78.88	153.03	93.28	95.44
500	92.68	184.13	111.69	115.84
Average	55.45	102.19	67.13	69.62

However, the computational times taken by the IBFA (shown in Table 14) is significantly less than that of the BFA. It is evident because in the BFA, all the bacteria keep moving in the chemotaxis step, while in the IBFA, each bacterium stops moving in the chemotaxis step when it reaches near-optimum zone. Thus, it reduces the exploration capability of the bacteria and reduces the unnecessary oscillatory movement of bacteria at optimum solution.

On the other hand, after using the NSGA-II for tuning the IBFA, the process parameters are taken as mutation operator during elimination–dispersal: Swap-Shift, $w_{at} = 0.2$, $w_{rp} = 5.05$, $p_{ed} = 0.95$, and % of closeness = 0.981. It is observed that the probability of elimination–dispersal (p_{ed}) increases from 0.5 to 0.95, which indicates the higher probability of each bacterium movement during elimination–dispersal step. Accordingly, the exploration capability of the tuned IBFA is more than that of the IBFA without tuning. Therefore, the solutions obtained by the IBFA after tuning are better than those of the IBFA without tuning in most of the problem instances as reported in Tables 12–13, however, at the cost of some additional computational times (as shown in Table 14) due to extra mutation operations during the elimination–dispersal step.

Similar to other compared metaheuristic algorithms, we considered the population size N = 20 for the GA1 [9]. However, it is observed from the results of Tables 15–17 that there is a tendency of stopping before reaching maximum number of iterations and the solution of the GA1 is further improved with N = 300 (as given in Carlier et al. [9]. This is evident due to the use of large population size of the GA1. As a result, the probability of being trapped into local optimum for the GA1 with N = 300 is much less than that of the GA1 with N = 20.

Tables 18–20 show the comparative results of the proposed method with those of GA1, SA and DCS with respect to the best, worst and mean solution considering each problem size for datasets 1, 2 and 3, respectively. The results of these tables reveal that the proposed method performs significantly better than GA1 in all problem instances for all datasets. However, the proposed method performs slightly better than the SA and DCS. However, for the problem size of 200 in dataset 1, the proposed method and the DCS show similar performance. Moreover, the SA and DCS are comparable to each other. The results also indicate that the proposed method outperforms the compared algorithms for all the problem instances of datasets 2 and 3. Thus, it is evident that the proposed method is quite robust to minimize the cycle time for the 2-machine RCS problem, especially in situations involving relatively high variability of the processing times, loading/unloading times and the transporting time of the robot.

Table 15
Results of the best, worst and mean *CT* values of various algorithms for Dataset 1.

No. of parts	GA1 (N = 300) (2010)			GA1 (N = 20) (2010)		
	Best	Worst	Mean	Best	Worst	Mean
200	35702	35728	35720	35742	35896	35825
225	39658	39795	39732	39798	39849	39832
250	44239	44280	44252	44344	44349	44346
275	48586	48655	48622	48691	48890	48787
300	53337	53484	53402	53585	53671	53614
325	57399	57428	57415	57588	57599	57594
350	62119	62185	62142	62119	62185	62142
375	66639	66700	66672	66690	66885	66782
400	70453	70499	70479	70579	70648	70625
425	75127	75240	75170	75292	75397	75360
450	80251	80386	80317	80418	80598	80516
475	84152	84261	84204	84481	84499	84492
500	89106	89161	89128	89284	89415	89343
Average	62059	62138	62096	62201	62299	62251

Table 16
Results of the percentage of deviation of various algorithms for Dataset 1.

No. of parts	*LB*	GA1 (N = 300) (2010)		GA1 (N = 20) (2010)	
		Best *CT*	*PD*	Best *CT*	*PD*
200	31278	35702	14.14	35742	14.27
225	34724	39658	14.20	39798	14.61
250	38569	44239	14.70	44344	14.97
275	42556	48586	14.17	48691	14.42
300	46758	53337	14.07	53585	14.60
325	50308	57399	14.09	57588	14.47
350	54374	62119	14.24	62119	14.24
375	58244	66639	14.41	66690	14.50
400	61446	70453	14.66	70579	14.86
425	65619	75127	14.49	75292	14.74
450	70216	80251	14.29	80418	14.53
475	73528	84152	14.45	84481	14.90
500	77561	89106	14.88	89284	15.11
Average	54245	62059	14.37	62200	14.63

Table 17
Average CPU time (in seconds) taken by various algorithms for Dataset 1.

No. of parts	GA1 (N = 300) (2010)	GA1 (N = 20) (2010)
200	55.90	39.89
225	86.55	44.14
250	90.40	53.79
275	95.66	61.26
300	105.60	70.63
325	108.21	79.69
350	113.92	86.31
375	121.49	98.83
400	122.40	115.33
425	146.77	128.14
450	159.01	129.62
475	205.65	142.36
500	245.73	154.09
Average	127.48	92.62

Tables 21–23 display the comparative results of the PD of the proposed method and the compared algorithms for Datasets 1, 2 and 3, respectively. Following Eq. (19), the PD for a given problem instance is computed as the percentage deviation of the LB and the best cycle time on that problem instance. Table 21 shows that the PD for the proposed method ranges from 13.25% to 14.10% compared with the corresponding values of 14.07% −14.88%, 13.27%–14.11%, and 13.27%–14.11% for the GA, SA and DCS, respectively. Clearly, the proposed method demonstrates its superior ability to solve large-sized problems. Moreover, based on the results, the proposed method outperforms the GA and it produces marginally better solution than the SA and DCS for Dataset 1. The results in Tables 22 and 23 reveal that the proposed method with respect to the PD performs best among the four

Table 18
Results of the best, worst and mean *CT* values of various algorithms for Dataset 1.

No. of parts	GA1 (2010)			SA (2016)			DCS (2016)			Proposed IBFA		
	Best	Worst	Mean	Best	Worst	Mean	Best	Worst	Mean	Best	Worst	Mean
200	35702	35728	35720	35540	35544	35542	35534	35535	35534	35534	35535	35534
225	39658	39795	39732	39499	39502	39500	39493	39497	39495	39492	39493	39492
250	44239	44280	44252	44013	44020	44017	44011	44016	44013	44010	44011	44010
275	48586	48655	48622	48321	48332	48326	48319	48326	48322	48317	48319	48318
300	53337	53484	53402	53033	53044	53038	53039	53042	53041	53031	53032	53031
325	57399	57428	57415	56982	56997	56991	56986	56987	56986	56975	56981	56977
350	62119	62185	62142	61712	61728	61717	61712	61763	61727	61707	61709	61707
375	66639	66700	66672	66178	66188	66184	66181	66212	66192	66168	66176	66172
400	70453	70499	70479	69929	69948	69939	69932	70053	69961	69924	69928	69926
425	75127	75240	75170	74576	74582	74578	74579	74648	74596	74565	74571	74567
450	80251	80386	80317	79700	79718	79707	79702	79722	79716	79692	79696	79693
475	84152	84261	84204	83486	83508	83499	83520	83528	83528	83481	83486	83483
500	89106	89161	89128	88433	88442	88436	88445	88468	88454	88427	88433	88429
Average	62059	62138	62096	61646	61657	61651	61650	61676	61658	61640	61643	61641

Table 19
Results of the best, worst and mean *CT* values of various algorithms for Dataset 2.

No. of parts	GA1 (2010)			SA (2016)			DCS (2016)			Proposed IBFA		
	Best	Worst	Mean	Best	Worst	Mean	Best	Worst	Mean	Best	Worst	Mean
200	92989	93237	93098	90646	91151	90910	90688	90851	90753	90311	90450	90371
225	105658	105935	105816	103288	103588	103443	102753	102974	102837	102476	102608	102576
250	116337	116514	116464	113480	113822	113647	111055	113744	111622	100609	100705	100673
275	128780	128991	128932	125655	126108	125848	125665	125875	125816	124931	125120	124997
300	141113	141431	141283	137761	138158	137906	137668	137989	137851	136988	137183	137065
325	152146	152379	152241	147808	148714	148339	148168	148354	148265	147441	147580	147532
350	185315	185690	185488	182756	182865	182814	182977	183014	182993	182562	182623	182592
375	187416	187613	187512	183653	184302	184045	184083	184235	184177	183545	183627	183601
400	198761	199242	198909	194521	195396	194827	194952	195270	195158	194122	194499	194418
425	211892	212406	212265	207795	208247	208030	207909	208283	208178	207682	207755	207731
450	224121	224743	224482	219867	220397	220040	220259	220430	220328	219553	219845	219713
475	236863	237248	237090	231923	232248	232014	232404	232793	232581	231544	231910	231735
500	249409	249726	249559	244537	244890	244658	245035	246076	245372	244190	244518	244402
Average	171600	171935	171780	167976	168453	168194	167970	168453	168149	166612	166802	166724

Table 20
Results of the best, worst and mean *CT* values of various algorithms for Dataset 3.

No. of parts	GA1(2010)			SA (2016)			DCS (2016)			Proposed IBFA		
	Best	Worst	Mean	Best	Worst	Mean	Best	Worst	Mean	Best	Worst	Mean
200	61479	61735	61591	60977	61173	61067	59859	60092	60006	59693	59764	59735
225	70380	70586	70527	69721	70182	69870	68382	68426	68401	68033	68117	68084
250	76806	76885	76839	75548	75854	75766	74556	74684	74649	74278	74311	74295
275	84715	85018	84809	85002	85561	85414	82317	82475	82403	81995	82038	82012
300	93699	93991	93839	92620	93280	93010	90879	90993	90964	90535	90621	90584
325	100935	101377	101134	99700	100076	99900	98151	98246	98195	97736	97751	97742
350	110092	110686	110357	108849	109267	109021	106987	107190	107121	106624	106723	106683
375	116548	116700	116625	116548	116700	116624	113038	113078	113054	112542	112690	112604
400	125233	126239	125758	124400	126796	125022	122068	122349	122145	121515	121549	121531
425	133393	133591	133483	131171	132039	131524	129175	129346	129243	128783	128830	128801
450	141407	142067	141745	138688	139530	139131	137324	137587	137509	136953	137056	136988
475	149720	150184	149990	147799	148577	148239	145483	146036	145655	144880	144994	144953
500	156727	156907	156815	153877	155016	154472	152134	152331	152218	151659	151701	151681
Average	109318	109690	109501	108069	108773	108389	106181	106372	106274	105787	105857	105823

algorithms and the DCS is the next best followed by SA and GA, respectively.

In this paper, the problem instances taken under consideration for the computational experiment are same as those used in Majumder and Laha [11]. Moreover, in both the present study and the study carried out by Majumder and Laha [11], a similar programming environment, i.e., MATLAB 2009a for coding as well as a similar PC configuration, i.e., Intel i5-2450M CPU with 4 GB RAM running at 2.50 GHz for execution of the code is used. Therefore, we reported the original results of the benchmark problems produced by the DCS [11]. However, since both GA1 (2010) and SA (2016) have used different programming languages and the PC configurations in the original literature, they were coded in MATLAB 2009a programming environment and executed on a PC with Intel i5-2450M CPU with 4 GB RAM running at 2.50 GHz. In addition, the problems considered in the present study are different from those considered in the original papers of the GA1 (2010) and SA (2016). As a result, the computational results of the GA1 and SA reported in the present study are different from the results given in the existing literature.

Moreover, in order to assess the performance of the proposed algorithm for solving large-sized RCS problems, two similar existing discrete BFAs, namely, basic-BFA [12] and chaotic local search-based (CLS-BFA) [12] were compared with the solutions given by the proposed algorithm. We considered the population size as 20 for both the existing BFAs and the input parameters

Table 21
Results of the percentage of deviation of various algorithms for Dataset 1.

No. of parts	*LB*	GA1 (2010)		SA (2016)		DCS (2016)		Proposed IBFA	
		Best *CT*	*PD*	Best *CT*	*PD*	Best *CT*	*PD*	Best *CT*	*PD*
200	31278	35702	14.14	35540	13.63	35534	13.60	35534	13.60
225	34724	39658	14.20	39499	13.75	39493	13.73	39492	13.73
250	38569	44239	14.70	44013	14.11	44011	14.11	44010	14.10
275	42556	48586	14.17	48321	13.55	48319	13.54	48317	13.53
300	46758	53337	14.07	53033	13.42	53039	13.43	53031	13.41
325	50308	57399	14.09	56982	13.27	56986	13.27	56975	13.25
350	54374	62119	14.24	61712	13.49	61712	13.49	61707	13.48
375	58244	66639	14.41	66178	13.62	66181	13.63	66168	13.60
400	61446	70453	14.66	69929	13.80	69932	13.81	69924	13.79
425	65619	75127	14.49	74576	13.65	74579	13.65	74565	13.63
450	70216	80251	14.29	79700	13.51	79702	13.51	79692	13.49
475	73528	84152	14.45	83486	13.54	83520	13.59	83481	13.53
500	77561	89106	14.88	88433	14.02	88445	14.03	88427	14.00
Average	54245	62059	14.37	61646	13.64	61650	13.65	61640	13.63

Table 22
Results of the percentage of deviation of various algorithms for Dataset 2.

No. of parts	*LB*	GA1 (2010)		SA (2016)		DCS (2016)		Proposed IBFA	
		Best *CT*	*PD*	Best *CT*	*PD*	Best *CT*	*PD*	Best *CT*	*PD*
200	73257	92989	26.94	90646	23.74	90688	23.79	90311	23.28
225	82472	105658	28.11	103288	25.24	102753	24.59	102476	24.26
250	91343	116337	27.36	113480	24.24	111055	21.58	100609	10.14
275	100442	128780	28.21	125655	25.10	125665	25.11	124931	24.38
300	109775	141113	28.55	137761	25.49	137668	25.41	136988	24.79
325	119005	152146	27.85	147808	24.20	148168	24.51	147441	23.89
350	154975	185315	19.58	182756	17.93	182977	18.07	182562	17.80
375	150729	187416	24.34	183653	21.84	184083	22.13	183545	21.77
400	160554	198761	23.80	194521	21.16	194952	21.42	194122	20.91
425	170946	211892	23.95	207795	21.56	207909	21.62	207682	21.49
450	181320	224121	23.61	219867	21.26	220259	21.48	219553	21.09
475	191424	236863	23.74	231923	21.16	232404	21.41	231544	20.96
500	200921	249409	24.13	244537	21.71	245035	21.96	244190	21.54
Average	137474	171600	25.40	167976	22.66	167970	22.54	166611	21.25

Table 23
Results of the percentage of deviation of various algorithms for Dataset 3.

No. of parts	LB	GA1 (2010)		SA (2016)		DCS (2016)		Proposed IBFA	
		Best *CT*	*PD*	Best *CT*	*PD*	Best *CT*	*PD*	Best *CT*	*PD*
200	54775	61479	12.24	60977	11.32	59859	9.28	59693	8.98
225	62677	70380	12.29	69721	11.24	68382	9.10	68033	8.55
250	68376	76806	12.33	75548	10.49	74556	9.04	74278	8.63
275	75788	84715	11.78	85002	12.16	82317	8.61	81995	8.19
300	83452	93699	12.28	92620	10.99	90879	8.90	90535	8.49
325	89958	100935	12.20	99700	10.83	98151	9.11	97736	8.65
350	98003	110092	12.34	108849	11.07	106987	9.17	106624	8.80
375	102900	116548	13.26	116548	13.26	113038	9.85	112542	9.37
400	111311	125233	12.51	124400	11.76	122068	9.66	121515	9.17
425	117209	133393	13.81	131171	11.91	129175	10.21	128783	9.87
450	125675	141407	12.52	138688	10.35	137324	9.27	136953	8.97
475	132716	149720	12.81	147799	11.36	145483	9.62	144880	9.17
500	138655	156727	13.03	153877	10.98	152134	9.72	151659	9.38
Average	97038	109318	12.57	108069	11.36	106181	9.35	105786	8.94

of these BFAs were taken from the paper of Zhao et al. [12]. The maximum number of iteration for all BFAs including the proposed algorithm was 800 and the corresponding n_c, n_r and n_{ed} values were set to 2, 100, and 4 respectively.

Tables 24–26 display the solutions obtained by the existing basic-BFA, CLS-BFA, DCS and the proposed algorithm for Datasets 1, 2 and 3. The results of these tables depict that the solution generated by the proposed algorithm is better than other algorithms. Further, Tables 27–29 show that the PD between the LB and the generated solutions of Basic-BFA, CLS-BFA, DCS and the proposed IBFA varies between 12.54–29.25, 11.67–28.84, 8.61–25.41 and 8.19–24.79 respectively, which clearly demonstrate the superiority of the proposed algorithm over other BFAs to produce optimal or near-optimal solutions for large-sized problems.

In case of the existing basic-BFA, CLS-BFA, the bacterium position was represented by the continuous variable during updating the position in chemotaxis strategy. However, a mechanism, called, smallest position value, was implemented to convert the position of bacterium from a continuous to a discrete variable. As a result, sometimes, it becomes very difficult to distinguish the sequence order of initial and final positions, which affects the motion of the bacterium and thus, the convergence rate becomes inconsistent. The inconsistency in the convergence can be observed in Fig. 3.

Table 24
Results of the best, worst and mean *CT* values of various algorithms for Dataset 1.

No. of Parts	Basic-BFA (2016)			CLS-BFA (2016)			DCS (2016)			Proposed IBFA		
	Best	Worst	Mean	Best	Worst	Mean	Best	Worst	Mean	Best	Worst	Mean
200	35780	35830	35802	35682	35755	35722	35534	35535	35534	35534	35535	35534
225	39822	39877	39850	39774	39804	39789	39493	39497	39495	39492	39493	39492
250	44372	44390	44377	44255	44308	44292	44011	44016	44013	44010	44011	44010
275	48697	48765	48738	48584	48671	48629	48319	48326	48322	48317	48319	48318
300	53506	53564	53535	53402	53441	53425	53039	53042	53041	53031	53032	53031
325	57497	57573	57534	57408	57470	57437	56986	56987	56986	56975	56981	56977
350	62305	62340	62323	62192	62263	62229	61712	61763	61727	61707	61709	61707
375	66823	66858	66842	66686	66761	66733	66181	66212	66192	66168	66176	66172
400	70579	70640	70604	70428	70555	70506	69932	70053	69961	69924	69928	69926
425	75318	75388	75342	75111	75256	75209	74579	74648	74596	74565	74571	74567
450	80472	80522	80499	80348	80415	80389	79702	79722	79716	79692	79696	79693
475	84346	84420	84391	84239	84284	84255	83520	83528	83528	83481	83486	83483
500	89237	89330	89273	89035	89211	89126	88445	88468	88454	88427	88433	88429
Average	62212	62269	62239	62088	62169	62134	61650	61676	61658	61640	61643	61641

Table 25
Results of the best, worst and mean *CT* values of various algorithms for Dataset 2.

No. of Parts	Basic-BFA (2016)			CLS-BFA (2016)			DCS (2016)			Proposed IBFA		
	Best	Worst	Mean	Best	Worst	Mean	Best	Worst	Mean	Best	Worst	Mean
200	93670	93900	93741	93096	93589	93289	90688	90851	90753	90311	90450	90371
225	106120	106360	106264	105611	106097	105857	102753	102974	102837	102476	102608	102576
250	116912	117125	117039	116401	116837	116579	111055	113744	111622	100609	100705	100673
275	129432	129700	129549	128911	129482	129195	125665	125875	125816	124931	125120	124997
300	141888	142130	141987	141431	141938	141603	137668	137989	137851	136988	137183	137065
325	152914	153161	153025	151995	152654	152369	148168	148354	148265	147441	147580	147532
350	185856	185987	185934	185432	185760	185603	182977	183014	182993	182562	182623	182592
375	188093	188387	188310	187390	188111	187820	184083	184235	184177	183545	183627	183601
400	199453	199828	199693	198871	199265	199078	194952	195270	195158	194122	194499	194418
425	212899	213103	213026	212006	212723	212468	207909	208283	208178	207682	207755	207731
450	225320	225414	225368	224495	224963	224805	220259	220430	220328	219553	219845	219713
475	237591	237886	237730	237020	237479	237230	232404	232793	232581	231544	231910	231735
500	250501	250590	250549	249850	250253	250031	245035	246076	245372	244190	244518	244402
Average	172358	172582	172478	171732	172242	171994	167970	168453	168149	166612	166802	166724

Table 26
Results of the best, worst and mean *CT* values of various algorithms for Dataset 3.

No. of Parts	Basic-BFA (2016)			CLS-BFA (2016)			DCS (2016)			Proposed IBFA		
	Best	Worst	Mean	Best	Worst	Mean	Best	Worst	Mean	Best	Worst	Mean
200	62044	62191	62111	61593	61829	61722	59859	60092	60006	59693	59764	59735
225	71016	71190	71102	70298	70860	70534	68382	68426	68401	68033	68117	68084
250	77208	77452	77352	76646	77175	76920	74556	74684	74649	74278	74311	74295
275	85293	85500	85392	84636	85269	84949	82317	82475	82403	81995	82038	82012
300	94349	94527	94458	93688	94200	94030	90879	90993	90964	90535	90621	90584
325	101582	101981	101791	101334	101459	101392	98151	98246	98195	97736	97751	97742
350	110914	111161	111022	110453	110820	110636	106987	107190	107121	106624	106723	106683
375	117315	117550	117438	116700	117034	116843	113038	113078	113054	112542	112690	112604
400	126643	126924	126773	125850	126351	126051	122068	122349	122145	121515	121549	121531
425	134176	134424	134325	133348	134002	133748	129175	129346	129243	128783	128830	128801
450	142267	142446	142346	141439	142276	141793	137324	137587	137509	136953	137056	136988
475	150684	150889	150789	149854	150626	150342	145483	146036	145655	144880	144994	144953
500	157376	157782	157567	156653	157048	156905	152134	152331	152218	151659	151701	151681
Average	110067	110309	110190	109423	109919	109682	106181	106372	106274	105787	105857	105823

On the other hand, based on the convergence time reported in Tables 30–32, it is evident that the proposed algorithm is the fastest one among the basic-BFA, CLS-BFA. Tables 30–32 show the average computational times for single execution taken by the proposed method and the compared algorithms for Datasets 1, 2 and 3, respectively. From the results of these tables, it is evident that the proposed method is the fastest among all compared algorithms and the SA is the slowest among them. Overall, the proposed algorithm requires 50%, 73% and 54% of the convergence times taken by the DCS, SA and GA1, respectively. On the other hand, the computational times required by GA1 and DCS are comparable. This is expected since the Dec_Ratio of the SA was very high (0.99) which makes the convergence rate slow and subsequently, the algorithm becomes slowest in speed. In case of SA, the number of iterations considered for each temperature was dependent on the job size of the particular problem. However, since the rate of change in temperature i.e., cooling rate taken under consideration is very low, it increases the overall number of iterations taken by the SA for convergence. This makes the algorithm slow, requiring more time by the algorithm. In addition, the GA1 takes more computational time due to large population size required for solving the problem, whereas, large step size at the initial stage slows down the computation speed of the DCS. The results of these tables also reveal that the computational time of the proposed IBFA is comparable to that of the Basic-BFA and both the proposed IBFA and the Basic-BFA are faster than the CLS-BFA.

Table 27
Results of the percentage of deviation of various algorithms for Dataset 1.

No. of Parts	Lower Bound	Basic-BFA (2016)		CLS-BFA (2016)		DCS (2016)		Proposed IBFA	
		Best *CT*	*PD*	Best *CT*	*PD*	Best *CT*	*PD*	Best *CT*	*PD*
200	31278	35780	14.39	35682	14.08	35534	13.60	35534	13.60
225	34724	39822	14.68	39774	14.54	39493	13.73	39492	13.73
250	38569	44372	15.04	44255	14.74	44011	14.11	44010	14.10
275	42556	48697	14.43	48584	14.16	48319	13.54	48317	13.53
300	46758	53506	14.43	53402	14.20	53039	13.43	53031	13.41
325	50308	57497	14.29	57408	14.11	56986	13.27	56975	13.25
350	54374	62305	14.58	62192	14.37	61712	13.49	61707	13.48
375	58244	66823	14.73	66686	14.49	66181	13.63	66168	13.60
400	61446	70579	14.86	70428	14.61	69932	13.81	69924	13.79
425	65619	75318	14.78	75111	14.46	74579	13.65	74565	13.63
450	70216	80472	14.60	80348	14.42	79702	13.51	79692	13.49
475	73528	84346	14.71	84239	14.56	83520	13.59	83481	13.53
500	77561	89237	15.05	89035	14.79	88445	14.03	88427	14.00
Average	54245	62212	14.66	62088	14.43	61650	13.65	61640	13.63

Table 28
Results of the percentage of deviation of various algorithms for Dataset 2.

No. of Parts	Lower Bound	Basic-BFA (2016)		CLS-BFA (2016)		DCS (2016)		Proposed IBFA	
		Best *CT*	*PD*	Best *CT*	*PD*	Best *CT*	*PD*	Best *CT*	*PD*
200	73257	93670	27.86	93096	27.08	90688	23.79	90311	23.28
225	82472	106120	28.67	105611	28.06	102753	24.59	102476	24.26
250	91343	116912	27.99	116401	27.43	111055	21.58	100609	10.14
275	100442	129432	28.86	128911	28.34	125665	25.11	124931	24.38
300	109775	141888	29.25	141431	28.84	137668	25.41	136988	24.79
325	119005	152914	28.49	151995	27.72	148168	24.51	147441	23.89
350	154975	185856	19.93	185432	19.65	182977	18.07	182562	17.80
375	150729	188093	24.79	187390	24.32	184083	22.13	183545	21.77
400	160554	199453	24.23	198871	23.87	194952	21.42	194122	20.91
425	170946	212899	24.54	212006	24.02	207909	21.62	207682	21.49
450	181320	225320	24.27	224495	23.81	220259	21.48	219553	21.09
475	191424	237591	24.12	237020	23.82	232404	21.41	231544	20.96
500	200921	250501	24.68	249850	24.35	245035	21.96	244190	21.54
Average	137474	172358	25.97	171732	25.48	167970	22.54	166611	21.25

Table 29
Results of the percentage of deviation of various algorithms for Dataset 3.

No. of Parts	Lower Bound	Basic-BFA (2016)		CLS-BFA (2016)		DCS (2016)		Proposed IBFA	
		Best *CT*	*PD*	Best *CT*	*PD*	Best *CT*	*PD*	Best *CT*	*PD*
200	54775	62044	13.27	61593	12.45	59859	9.28	59693	8.98
225	62677	71016	13.30	70298	12.16	68382	9.10	68033	8.55
250	68376	77208	12.92	76646	12.09	74556	9.04	74278	8.63
275	75788	85293	12.54	84636	11.67	82317	8.61	81995	8.19
300	83452	94349	13.06	93688	12.27	90879	8.90	90535	8.49
325	89958	101582	12.92	101334	12.65	98151	9.11	97736	8.65
350	98003	110914	13.17	110453	12.70	106987	9.17	106624	8.80
375	102900	117315	14.01	116700	13.41	113038	9.85	112542	9.37
400	111311	126643	13.77	125850	13.06	122068	9.66	121515	9.17
425	117209	134176	14.48	133348	13.77	129175	10.21	128783	9.87
450	125675	142267	13.20	141439	12.54	137324	9.27	136953	8.97
475	132716	150684	13.54	149854	12.91	145483	9.62	144880	9.17
500	138655	157376	13.50	156653	12.98	152134	9.72	151659	9.38
Average	97038	110067	13.36	109423	12.67	106181	9.35	105786	8.94

On the other hand, the adaptive technique used in determining step size helps the proposed algorithm to reduce the unnecessary oscillatory movement of each bacterium at a local optimum or global optimum point and thus, it increases the convergence speed and subsequently, reduces the computational time taken by the algorithm during the convergence. However, when the step size becomes zero near the optimum point, the mutation operation used in the elimination–dispersal step helps the bacteria to move towards optimum point. Fig. 5 shows the rate of change in solution as well as change in value of the step size of the best bacterium with respect to the number of iterations.

We provided some convergence graphs to illustrate the behavior and performance of the proposed algorithm. Fig. 3 shows the convergence characteristics of the existing algorithms as well as the proposed algorithm. However, due to the variation in tuning the parameters and different stopping criteria, the maximum number of iterations taken by the compared algorithms differ largely and therefore, we considered the convergence of all the algorithms up to 1000 iterations. From Fig. 3, it reveals that the convergence rate of the proposed algorithm is consistent and fastest among the algorithms used in this study. On the other hand, a partial inconsistency in the convergence is observed for the GA1 and SA. For the Basic-BFA and CLS-BFA algorithms, the inconsistency in the convergence is significant. However, the convergence rate of the DCS and the SA is consistent but slow as compared to the proposed algorithm. The Fig. 3 also indicates

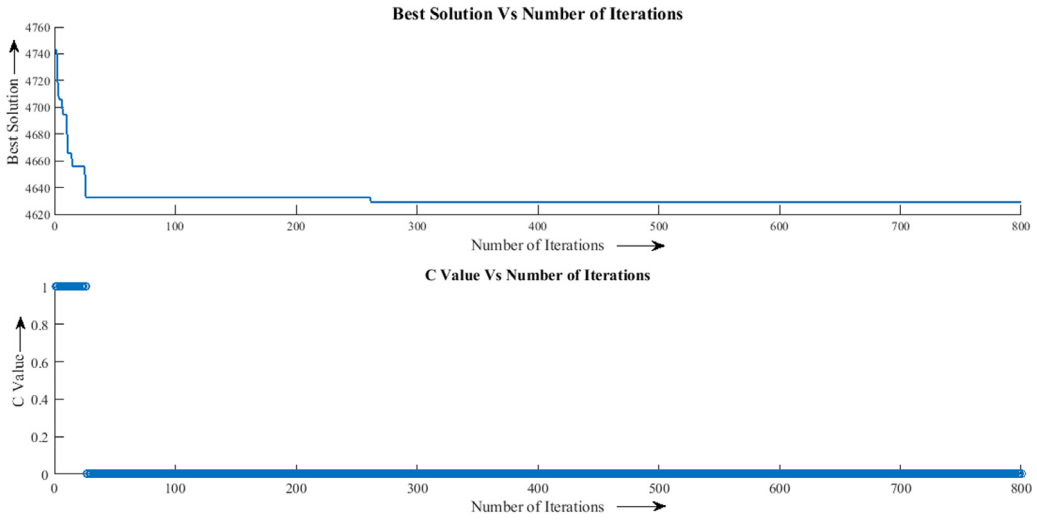

Fig. 5. Convergence of solution and change in C value with respect to number of iterations.

Table 30
Average CPU time (in seconds) taken by various algorithms for Dataset 1.

No. of parts	GA1 (2010)	SA (2016)	DCS (2016)	Basic-BFA (2016)	CLS-BFA (2016)	Proposed IBFA
200	55.90	164.36	60.14	32.04	124.45	24.31
225	86.55	165.84	82.75	44.86	135.76	26.81
250	90.40	175.20	87.78	45.13	140.14	28.42
275	95.66	183.75	104.26	48.65	155.56	29.87
300	105.60	199.21	106.45	49.72	186.17	31.46
325	108.21	200.27	106.78	50.06	208.77	34.92
350	113.92	202.80	111.21	58.07	216.21	93.97
375	121.49	205.57	126.76	61.66	254.18	103.64
400	122.40	209.85	128.05	69.56	265.82	108.56
425	146.77	241.31	132.89	71.37	283.90	114.89
450	159.01	277.82	147.88	84.37	299.40	124.16
475	205.65	287.19	163.30	89.30	310.23	127.15
500	245.73	331.60	193.52	95.84	334.33	138.14
Average	127.48	218.83	119.37	61.58	224.22	75.87

Table 31
Average CPU time (in seconds) taken by various algorithms for Dataset 2.

No. of parts	GA1 (2010)	SA (2016)	DCS (2016)	Basic-BFA (2016)	CLS-BFA (2016)	Proposed IBFA
200	84.16	182.08	92.63	31.44	69.45	41.5
225	93.38	198.31	101.09	33.68	88.37	48.63
250	129.06	215.33	105.39	40.74	92.44	50.87
275	139.22	214.58	136.09	40.85	114.93	53.47
300	164.95	253.97	160.12	42.03	122.40	56.21
325	184.57	256.57	183.65	50.52	126.71	60.52
350	203.61	294.33	184.20	58.32	141.33	65.70
375	213.87	296.97	199.28	60.20	143.72	70.21
400	222.44	308.60	223.49	61.92	146.82	76.14
425	243.95	308.29	230.50	63.65	169.36	83.62
450	260.95	347.64	248.03	65.95	171.05	86.94
475	266.22	425.27	260.98	78.88	174.14	95.44
500	278.51	431.78	291.17	92.68	200.29	115.84
Average	191.14	287.21	185.9	55.45	135.46	69.62

that the mutation operations used in the proposed algorithm enhances the convergence rate of the proposed algorithm sharply.

Therefore, the above computational results demonstrate that the overall performance of the proposed algorithm is relatively more effective and efficient than the state-of-the-art algorithms. We have implemented some modifications in the proposed IBFA to enhance the results of the basic BFA. First, the mutation operation was used in the chemotaxis step to make the bacterium motion distinct. Second, the adaptive chemotaxis step size C considered in the IBFA reduces the convergence time of the proposed algorithm. Third, the mutation operation was used during the elimination–dispersal strategy for the bacterium to avoid being stuck in local optimum. Moreover, the tuning of the process parameters through the RSM-NSGA-II-Desirability technique helps the proposed algorithm to achieve a balance between the exploration and the exploitation.

Table 32
Average CPU time (in seconds) taken by various algorithms for Dataset 3.

No. of parts	GA1 (2010)	SA (2016)	DCS (2016)	Basic-BFA (2016)	CLS-BFA (2016)	Proposed IBFA
200	82.03	163.56	72.93	24.43	67.21	26.17
225	82.70	182.60	75.74	29.63	102.47	29.07
250	93.96	210.25	85.94	32.58	104.69	34.12
275	97.07	230.22	101.07	34.94	105.99	48.88
300	102.52	242.54	123.65	38.38	113.01	50.52
325	113.79	242.06	125.32	39.83	113.43	63.87
350	132.12	262.60	127.45	43.94	123.47	78.40
375	154.43	295.07	137.79	48.47	159.64	87.86
400	179.41	320.16	148.86	51.62	170.31	108.56
425	253.65	361.46	149.16	52.85	202.36	118.24
450	351.03	407.22	175.67	60.15	237.73	124.49
475	385.74	447.57	241.47	62.86	241.71	129.36
500	483.33	467.42	299.14	82.80	257.89	134.25
Average	193.21	294.82	143.40	46.35	153.84	79.52

7. Conclusion

This paper considers a two-machine robotic cell scheduling problem with one-unit cycle, sequence-dependent setup times and different loading/unloading times of the parts. We have presented a discrete bacterial foraging algorithm to solve this problem with the objective of minimizing the cycle time. To implement the discrete version of this algorithm, the pairwise interchange mutation was chosen for tumbling and swimming operations during chemotaxis, while the cyclic shift neighborhood mutation was used for random movement during elimination–dispersal step. The exhaustive computational results considering a set of benchmark large-sized problems reveal that the proposed algorithm outperforms the existing state-of-the-art algorithms, while retaining its comparable average CPU time.

We recommend further research in the use of the bacterial foraging algorithm to solve a variety of robotic cell problems. In addition, hybrid metaheuristic algorithms as well as quantum based bacterial foraging algorithms have great potential for developing efficient optimization algorithms to solve various problems.

Acknowledgments

We thank the reviewers for their detailed constructive comments on earlier versions of the draft to considerable improve the quality and presentation of the paper.

References

[1] E. Levner, V. Kats, D.A.L. De Pablo, Cyclic scheduling in robotic cells: an extension of basic models in machine scheduling theory, in: Eugene Levner (Ed.), Multiprocessor Scheduling Theory and Applications, Itech Education and Publishing, Vienna, Austria, 2007, pp. 1–20.
[2] M. Dawande, H.N. Geismar, S.P. Sethi, C. Sriskandarajah, Sequencing and scheduling in robotic cells: Recent developments, J. Sched. 8 (5) (2005) 387–426.
[3] H. Kamoun, N.G. Hall, C. Sriskandarajah, Scheduling in robotic cells: Heuristics and cell design, Oper. Res. 47 (6) (1999) 821–835.
[4] M.G. Nejad, H. Güden, B. Vizvári, R.V. Barenji, A mathematical model and simulated annealing algorithm for solving the cyclic scheduling problem of a flexible robotic cell, Adv. Mech. Eng. 10 (1) (2018) 1687814017753912.
[5] A. Al-Ahmari, Optimal robotic cell scheduling with controllers using mathematically based timed Petri nets, Inform. Sci. 329 (2016) 638–648.
[6] S.Q. Liu, E. Kozan, A hybrid metaheuristic algorithm to optimize a real-world robotic cell, Comput. Oper. Res. 84 (2017) 188–194.
[7] A. Elmi, S. Topaloglu, Cyclic job shop robotic cell scheduling problem: Ant colony optimization, Comput. Ind. Eng. 111 (2017) 417–432.
[8] M.F. Zarandi, H. Mosadegh, M. Fattahi, Two-machine robotic cell scheduling problem with sequence-dependent setup times, Comput. Oper. Res. 40 (5) (2013) 1420–1434.
[9] J. Carlier, M. Haouari, M. Kharbeche, A. Moukrim, An optimization-based heuristic for the robotic cell problem, European J. Oper. Res. 202 (3) (2010) 636–645.
[10] G.D. Batur, S. Erol, O.E. Karasan, Robot move sequence determining and multiple part-type scheduling in hybrid flexible flow shop robotic cells, Comput. Ind. Eng. 100 (2016) (2016) 72–87.
[11] A. Majumder, D. Laha, A new cuckoo search algorithm for 2-machine robotic cell scheduling problem with sequence-dependent setup times, Swarm Evol. Comput. 28 (2016) 131–143.
[12] F. Zhao, Y. Liu, Z. Shao, X. Jiang, C. Zhang, J. Wang, A chaotic local search based bacterial foraging algorithm and its application to a permutation flow-shop scheduling problem, Int. J. Comput. Integr. Manuf. 29 (9) (2016) 962–981.
[13] L.W. Phillips, P.S. Unger, Mathematical programming solution of a hoist scheduling program, AIIE Trans. 8 (2) (1976) 219–225.
[14] J. Liu, Y. Jiang, Z. Zhou, Cyclic scheduling of a single hoist in extended electroplating lines: a comprehensive integer programming solution, IIE Trans. 34 (2002) 905–914.
[15] J. Leung, G. Zhang, Optimal cyclic scheduling for printed circuit board production lines with multiple hoists and general processing sequence, IEEE Trans. Robot. Autom. 19 (2003) 480–484.
[16] A. El Amraoui, M.A. Manier, A. El Moudni, M. Benrejeb, A mixed linear program for a multi-part cyclic hoist scheduling problem, in: Special issue on CEM, Int. J. Sci. Tech. Autom. Control Comput. Eng. 2 (2008) 612–623.
[17] A. Che, C. Chu, Cyclic hoist scheduling in large real-life electroplating lines, OR Spectrum 29 (2007) 445–470.
[18] Lei L., Q. Liu, Optimal cyclic scheduling of a robotic processing line with two-product and time-window constraints, INFOR Inf. Syst. Oper. Res. 39 (2001) 185–199.
[19] A. Che, Z. Zhou, C. Chu, H. Chen, Multi-degree cyclic hoist scheduling with time window constraints, Int. J. Prod. Res. 49 (2011) 5679–5693.
[20] L. Lei, R. Armstrong, S. Gu, Minimizing the fleet size with dependent time-window and single-track constraints, Oper. Res. Lett. 14 (1993) 91–98.
[21] Y. Yih, An algorithm for hoist scheduling problems, Int. J. Prod. Res. 32 (1994) 501–516.
[22] G.D. Batur, O.E. Karasan, M.S. Akturk, Multiple part-type scheduling in flexible robotic cells, Int. J. Prod. Econ. 135 (2) (2012) 726–740.
[23] I. Nielsen, Q.V. Dang, G. Bocewicz, Z. Banaszak, A methodology for implementation of mobile robot in adaptive manufacturing environments, J. Intell. Manuf. (2015) 1–18, http://dx.doi.org/10.1007/s10845-015-1072-2.
[24] M.M.S. Abdulkader, M.M. ElBeheiry, N.H. Afia, A.K. El-Kharbotly, Scheduling and sequencing in four machines robotic cell: Application of genetic algorithm and enumeration techniques, Ain Shams Eng. J. 4 (3) (2013) 465–474.
[25] S.S. Zabihzadeh, J. Rezaeian, Two meta-heuristic algorithms for flexible flow shop scheduling problem with robotic transportation and release time, Appl. Soft Comput. 40 (2015) 319–330.
[26] I.N. Kamalabadi, A.H. Mirzaei, S. Gholami, A new hybrid particle swarm optimization algorithm to the cyclic multiple-part type three-machine robotic cell problem, in: Ester Martinez Martin (Ed.), Swarm Robotics from Biology to Robotics, INTECH Open Access Publisher, 2010, pp. 27–46.
[27] I.N. Kamalabadi, S. Gholami, A.H. Mirzaei, Considering a cyclic multiple-part type three-machine robotic cell problem, in: Industrial Engineering and Engineering Management, 2007 IEEE International Conference on Industrial Engineering and Engineering Management, Singapore, 2007, pp. 704–708.
[28] Q.K. Pan, M.F. Tasgetiren, Y.C. Liang, A discrete differential evolution algorithm for the permutation flowshop scheduling problem, Comput. Ind. Eng. 55 (4) (2008) 795–816.
[29] G. Deng, X. Gu, A hybrid discrete differential evolution algorithm for the no-idle permutation flow shop scheduling problem with makespan criterion, Comput. Oper. Res. 39 (9) (2012) 2152–2160.
[30] S. Zhou, M. Liu, H. Chen, X. Li, An effective discrete differential evolution algorithm for scheduling uniform parallel batch processing machines with non-identical capacities and arbitrary job sizes, Int. J. Prod. Econ. 179 (2016) 1–11.
[31] K.M. Passino, Biomimicry of bacterial foraging for distributed optimization and control, IEEE Control Syst. Mag. 22 (3) (2002) 52–67.
[32] B. Panigrahi, V.R. Pandi, S. Das, S. Das, Multiobjective fuzzy dominance based bacterial foraging algorithm to solve economic emission dispatch problem, Energy 35 (2011) 4761–4770.
[33] N. Pandit, A. Tripathi, S. Tapaswi, M. Pandit, An improved bacterial foraging algorithm for combined static/dynamic environmental economic dispatch, Appl. Soft Comput. 1 (2012) 3500–3513.
[34] P.D. Sathya, R. Kayalvizhi, Optimal multilevel thresholding using bacterial foraging algorithm, Expert Syst. Appl. 1 (38) (2011) 15549–15564.
[35] S. Kamyab, A. Bahrololoum, Designing of rule base for a tsk-fuzzy system using bacterial foraging optimization algorithm, Soc. Behav. Sci. 1 (2012) 176–183.
[36] H. Chen, Y. Zhu, K. Hu, Cooperative bacterial foraging optimization, Discrete Dyn. Nat. Soc. (2009) http://dx.doi.org/10.1155/2009/815247.
[37] S.K. Panda, S. Padhee, A. Kumar, S.S. Mahapatra, Optimization of fused deposition modelling (FDM) process parameters using bacterial foraging technique, Intell. Inf. Manag. 1 (02) (2009) 89.
[38] H. Nouri, T.S. Hong, A bacteria foraging algorithm based cell formation considering operation time, J. Manuf. Syst. 31 (3) (2012) 326–336.
[39] C. Liu, J. Wang, J.Y.T. Leung, K. Li, Solving cell formation and task scheduling in cellular manufacturing system by discrete bacterial foraging algorithm, Int. J. Prod. Res. 54 (3) (2015) 923–944.
[40] S. Dasgupta, S. Das, A. Biswas, A. Abraham, Automatic circle detection on digital images with an adaptive bacterial foraging algorithm, Soft Comput. 14 (11) (2010) 1151–1164.
[41] E. Bermejo, O. Cordón, S. Damas, J. Santamaría, A comparative study on the application of advanced bacterial foraging models to image registration, Inform. Sci. 295 (2015) 160–181.
[42] V.P. Sakthivel, R. Bhuvaneswari, S. Subramanian, Bacterial foraging technique based parameter estimation of induction motor from manufacturer data, Electr. Power Compon. Syst. 38 (6) (2010) 657–674.
[43] C. Wu, N. Zhang, J. Jiang, J. Yang, Y. Liang, Improved bacterial foraging algorithms and their applications to job shop scheduling problems, Adaptive and Natural Computing Algorithms: 8th International Conference, ICANNGA 2007, Warsaw, Poland, 2007, pp. 562–569.
[44] I. Chana, Bacterial foraging based hyper-heuristic for resource scheduling in grid computing, Future Gener. Comput. Syst. 29 (2013) 751–762.
[45] S. Devi, M. Geethanjali, Application of modified bacterial foraging optimization algorithm for optimal placement and sizing of distributed generation, Expert Syst. Appl. 41 (2014) 2772–2781.
[46] C. Liu, J. Wang, J.Y.T. Leung, K. Li, Solving cell formation and task scheduling in cellular manufacturing system by discrete bacteria foraging algorithm, Int. J. Prod. Res. 54 (3) (2016) 923–944.
[47] C. Liu, J. Wang, J.Y.T. Leung, Worker assignment and production planning with learning and forgetting in manufacturing cells by hybrid bacteria foraging algorithm, Comput. Ind. Eng. 96 (2016) 162–179.
[48] C. Liu, J. Wang, J.Y.T. Leung, Integrated bacteria foraging algorithm for cellular manufacturing in supply chain considering facility transfer and production planning, Appl. Soft Comput. 62 (2018) 602–618.
[49] Y. Atasagun, Y. Kara, Bacterial foraging optimization algorithm for assembly line balancing, Neural Comput. Appl. 25 (1) (2014) 237–250.
[50] S. Prakash, D.P. Vidyarthi, A hybrid GABFO scheduling for optimal makespan in computational grid, Int. J. Appl. Evol. Comput. 5 (3) (2014) 57–83.
[51] H. Nouri, S.H. Tang, A bacteria foraging algorithm based cell formation considering operation time, J. Manuf. Syst. 31 (3) (2012) 326–336.
[52] H. Nouri, S.H. Tang, Development of bacteria foraging optimization algorithm for cell formation in cellular manufacturing system considering cell load variations, J. Manuf. Syst. 32 (2013) 20–31.
[53] H. Nouri, S.H. Tang, M.K. Tuah, BASE: A Bacteria foraging algorithm for cell formation with sequence data, J. Manuf. Syst. 29 (2) (2010) 102–110.

[54] H. Nouri, Development of a comprehensive model and BFO algorithm for a dynamic cellular manufacturing system, Appl. Math. Model. 40 (2016) 1514–1531.
[55] H. Chen, Y. Zhu, K. Hu, L. Ma, Bacterial colony foraging algorithm: Combining chemotaxis, cell-to-cell communication, and self-adaptive strategy, Inform. Sci. 273 (2014) 73–100.
[56] S. Dasgupta, S. Das, A. Abraham, A. Biswas, Adaptive computational chemotaxis in bacterial foraging optimization: an analysis, IEEE Trans. Evol. Comput. 13 (4) (2009) 919–941.
[57] K. Deb, A. Pratap, S. Agarwal, T.A.M.T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II, IEEE Trans. Evol. Comput. 6 (2) (2002) 182–197.
[58] N. Delgarm, B. Sajadi, S. Delgarm, F. Kowsary, A novel approach for the simulation-based optimization of the buildings energy consumption using NSGA-II: Case study in Iran, Energy Build. 127 (2016) 552–560.
[59] N. Fallah, S. Honarparast, NSGA-II Based multi-objective optimization in design of Pall friction dampers, J. Construct. Steel Res. 89 (2013) 75–85.