Expert Systems with Applications 40 (2013) 7231–7241

Contents lists available at SciVerse ScienceDirect

Expert Systems with Applications

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

Artificial bee colony algorithm for solving sequence-dependent disassembly line balancing problem

Can B. Kalayci [a,1], Surendra M. Gupta [b,*]

[a] Department of Industrial Engineering, Pamukkale University, Kinikli Kampusu, 20070 Denizli, Turkey
[b] 334 SN, Department of Mechanical and Industrial Engineering, Northeastern University, 360 Huntington Avenue, Boston, MA 02115, USA

ARTICLE INFO

Keywords:
Product recovery
Disassembly
Sequence-dependent disassembly line balancing
Combinatorial optimization
Metaheuristics
Artificial bee colony

ABSTRACT

In this paper, we consider a sequence-dependent disassembly line balancing problem (SDDLBP) with multiple objectives that requires the assignment of disassembly tasks to a set of ordered disassembly workstations while satisfying the disassembly precedence constraints and optimizing the effectiveness of several measures. Since the complexity of SDDLBP increases with the number of parts of the product, an efficient methodology based on artificial bee colony (ABC) is proposed to solve the SDDLBP. ABC is an optimization technique which is inspired by the behavior of honey bees. The performance of the proposed algorithm was tested against six other algorithms. The results show that the proposed ABC algorithm performs well and is superior to the other six algorithms in terms of the objective values performance.

1. Introduction

In green and reverse supply chains, environmental laws and directives are as important as cost minimization and profit maximization. See Wang and Gupta (2011) for more information on green supply chain management and Gupta (2013) for information on reverse supply chains. In reverse supply chains, product recovery is achieved for an end-of-life (EOL) product through recycling, refurbishing or remanufacturing depending on its condition. Gungor and Gupta (1999) and Ilgin and Gupta (2010) provide extensive reviews of product recovery. In remanufacturing, labor, energy and material used in the manufacturing process can be recovered since the returned products preserve their current form. However, in recycling, returned products are simply shredded. In other words, only material content of a returned product can be recovered. In refurbishment/repair, a returned product is kept functional by changing and/or repairing some components. In this case, the resultant product cannot be treated as a brand new product. However, remanufactured products often come with warranties similar to brand new products since they are completely reassembled using disassembled, upgraded or brand-new components. Therefore, remanufacturing is the most environment friendly and profitable product recovery option. See a recent book by Ilgin and Gupta (2012) for more information on remanufacturing.

The most critical and time consuming step of remanufacturing is disassembly. Disassembly tasks are often carried out using manual labor. However, firms have started to convert to automated disassembly systems at an increasing pace in order to deal with the rise in the amount of EOL products as well as to achieve higher productivity levels and reduced labor costs. Due to its high productivity and suitability for automation, disassembly line is the most suitable layout for disassembly operations (Gungor & Gupta, 2001). A disassembly line must be balanced to optimize the use of resources.

Disassembly operations have unique characteristics and cannot be considered as the reverse of assembly operations. The quality and quantity of components used in the stations of an assembly line can be controlled by imposing strict conditions. However, there are no such conditions of EOL products moving on a disassembly line. In a disassembly environment, the flow process is divergent; a single product is broken down into many subassemblies and parts while the flow process is convergent in an assembly environment. There could also be a high degree of uncertainty in the structure, quality, reliability and the condition of the returned products in disassembly. Additionally, some parts of the product may be hazardous and may require special handling that will affect the utilization of disassembly workstations. Since disassembly tends to be expensive, disassembly line balancing becomes significant in minimizing resources invested in disassembly and maximizing the level of automation.

Disassembly Line Balancing Problem (DLBP) is a multi-objective problem as described by Gungor and Gupta (2002) and has been mathematically proven to be NP-complete by McGovern and Gupta

* Corresponding author. Tel.: +1 (617) 373 4846; fax: +1 (617) 373 2921.
E-mail addresses: cbkalayci@pau.edu.tr (C.B. Kalayci), gupta@neu.edu (S.M. Gupta).
[1] Tel.: +90 (258) 296 3209; fax: +90 (258) 296 3262.

0957-4174/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.eswa.2013.06.067

(2007) making the goal to achieve the optimal balance computationally expensive. NP-complete or NP-hard are ways of showing that certain classes of problems are not solvable in realistic time (Tovey, 2002). Exhaustive search works well in obtaining optimal solutions for small sized instances. However, its exponential time complexity limits its application to large sized instances. An efficient search method, thus, needs to be employed to attain a (near) optimal condition with respect to objective functions. Although some researchers have formulated the DLBP using mathematical programming techniques (Altekin & Akkan, 2012; Altekin, Kandiller, & Ozdemirel, 2008; Koc, Sabuncuoglu, & Erel, 2009), it quickly becomes unsolvable for a practical sized problem due to its combinatorial nature. For this reason, there is an increasing need to use metaheuristic techniques such as genetic algorithms (GA) (Che, Che, & Hsu, 2009; Go, Wahab, Rahman, Ramli, & Hussain, 2012; Kalayci & Gupta, 2011a; McGovern & Gupta, 2007; Wang, Che, & Chiang, 2012), ant colony optimization (ACO) (Agrawal & Tiwari, 2008; Ding, Feng, Tan, & Gao, 2010; McGovern & Gupta, 2006), simulated annealing (SA) (Kalayci, Gupta, & Nakashima, 2012), tabu search (TS) (Kalayci & Gupta, 2011b), artificial bee colony (ABC) (Kalayci, Gupta, & Nakashima, 2011) and particle swarm optimization (PSO) (Kalayci & Gupta, 2012). See McGovern and Gupta (2011) for more information on DLBP. Sequence-dependency concept was introduced by Scholl, Boysen, and Fliedner (2006) to assembly line balancing literature while it was adapted to disassembly lines by (Kalayci & Gupta, 2013). In this paper, we consider a sequence-dependent disassembly line balancing problem (SDDLBP) with multiple objectives that requires the assignment of disassembly tasks to a set of ordered disassembly workstations while satisfying the disassembly precedence constraints and optimizing the effectiveness of several measures considering sequence-dependent time increments. Metaheuristic approaches are widely used to solve combinatorial optimization problems. An artificial bee colony algorithm is proposed to solve the SDDLBP and comparison with six metaheuristic approaches from the literature is provided.

The rest of the paper is organized as follows: In Section 2, notation used in this paper is presented. Problem definition and formulation are given in Section 3. Section 4 describes the proposed ABC algorithm for the multi-objective SDDLBP. The computational experience to evaluate its performance on numerical examples and the comparisons are provided in Section 5. Finally some conclusions are given in Section 6.

2. Notation

b	Bee count $(1,\ldots,eb)$
c	Cycle time (Maximum time available at each workstation)
d_i	Demand; quantity of part i requested
eb	Number of employed bees
F_e	Fitness values vector for employed bees
F_s	Fitness values vector for scout bees
F_o	Fitness values vector for onlooker bees
h_i	Binary value; 1 if part i is hazardous, else 0
i	Part identification, task count $(1,\ldots,n)$
$ilimit$	Iteration limit to call scout bees for exploration
IP	Set (i, j) of parts such that task i must precede task j
j	Part identification, task count $(1,\ldots,n)$
k	Workstation count $(1,\ldots,m)$
m	Number of workstations required for a given solution sequence
m^*	Minimum possible number of workstations
M	Sufficiently large number
n	Number of parts for removal
N	The set of natural numbers
ob	Number of onlooker bees
ps	Population size
PS_i	*ith* part in a solution sequence
r	Uniformly distributed random number between 0 and 1.
ric	Rest iteration count (The counter variable used to check release time of scout bees)
sb	Number of scout bees
sd_{ij}	Sequence dependent time increment influence of i on j
t_i	Part removal time of part i
t'_i	Part removal time of part i considering sequence dependent time increment
$time$	Time count for the algorithm
$tlimit$	Time limit of the algorithm to be executed

3. Problem definition and formulation

The sequence dependent disassembly line balancing problem investigated in this paper is concerned with a paced disassembly line for a single model of product that undergoes complete disassembly. Model assumptions include the following: a single product type is to be disassembled on a disassembly line; the supply of the end-of-life product is infinite; the exact quantity of each part available in the product is known and constant; a disassembly task cannot be divided between two workstations; each part has an assumed associated resale value which includes its market value and recycled material value; the line is paced; part removal times are deterministic, constant, and discrete; each product undergoes complete disassembly even if the demand is zero; all products contain all parts with no additions, deletions, modifications or physical defects; each part is assigned to one and only one workstation; the sum of part removal times of all the parts assigned to a workstation must not exceed cycle time; the precedence relationships among the parts must not be violated.

The difference between regular disassembly line balancing problem (DLBP) and sequence-dependent disassembly line balancing problem (SDDLBP) lies in the task time interactions. Task time interactions between two tasks occur when they do not have any precedence relationship, thus providing a choice in the order in which they may be performed. In a regular disassembly line balancing problem, the sequence in which tasks without precedence relationship between them are performed is not important. However, when sequence-dependent setup times between tasks are present in a disassembly line, the sequence in which precedence independent tasks are performed may matter. Thus, in SDDLBP, whenever precedence free tasks interact, their task times may be influenced based on the order in which they are performed. This may happen because one component may hinder the other component and thus may require additional movements and/or prevent it from using the most efficient or convenient disassembly process. Simply stated, for precedence free sequence dependent tasks i and j, disassembly time of task j is affected by task i if task i is yet to be performed in the disassembly sequence. Thus, if a precedence free sequence dependent task j is assigned to be done before task i is finished, sd_{ij} time units has be added to compute the total processing time of task j.

Illustrative example: The precedence relationships (solid line arrows) and sequence dependent time increments (dashed line arrows) for an 8 part PC disassembly process are illustrated in

Fig. 1 and their knowledge database is given in Table 1. Additional data required for sequence dependencies are as follows: $sd_{23} = 2$, $sd_{32} = 4$, $sd_{56} = 1$, $sd_{65} = 3$. This example is modified from Gungor and Gupta (2002).

For a feasible sequence ⟨1, 3, 2, 5, 6, 8, 7, 4⟩, since part 3 is disassembled before part 2, sequence dependency $sd_{23} = 2$ takes place because when part 3 is disassembled, the obstructing part 2 is still not taken out, i.e., the part removal time for part 3 ($t_3 = 12$) is increased which results in $t'_3 = t_3 + sd_{23} = 14$. Similarly, since part 5 is disassembled before part 6, sequence dependency $sd_{65} = 3$ takes place because when part 5 is disassembled, the obstructing part 6 is still not taken out, i.e., the part removal time for part 5 ($t_5 = 23$) is increased which results in $t'_5 = t_5 + sd_{65} = 26$.

In this paper, the precedence relationships considered are of AND type and are represented using the immediately preceding matrix $[y_{ij}]_{n\times n}$, where

$$y_{ij} = \begin{cases} 1 & \text{if task } i \text{ is executed after task } j \\ 0 & \text{if task } i \text{ is executed before task } j \end{cases} \tag{1}$$

In order to state the partition of total tasks, we use the assignment matrix $[x_{jk}]_{n\times m}$, where

$$x_{jk} = \begin{cases} 1 & \text{if part } j \text{ is assigned to station } k \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

The matrix $[sd_{ij}]_{n\times n}$ holds the sequence-dependent time increments data, where

$$sd_{ij} = \begin{cases} sd_{ij} & \text{if part } i \text{ prolongs removal time of part } j \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

The mathematical formulation of SDDLBP is given as follows:

$$\min f_1 = m \tag{4}$$

$$\min f_2 = \sum_{i=1}^{m} (c - t'_i)^2 \tag{5}$$

$$\min f_3 = \sum_{i=1}^{n} i \times h_{PS_i}, \quad h_{PS_i} = \begin{cases} 1 & \text{hazardous} \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

$$\min f_4 = \sum_{i=1}^{n} i \times d_{PS_i}, \quad d_{PS_i} \in N, \ \forall PS_i \tag{7}$$

Subject to:

$$\sum_{k=1}^{m} x_{jk} = 1, \quad j = 1, \ldots, n \tag{8}$$

$$\left\lceil \frac{\sum_{i=1}^{n} t_i}{c} \right\rceil \leqslant m^* \leqslant n \tag{9}$$

$$\sum_{j=1}^{n} \left(t_j + \sum_{i=1}^{n} sd_{ij} \times y_{ij} \right) \times x_{jk} \leqslant c \tag{10}$$

$$x_{ik} \leqslant \sum_{k=1}^{m} x_{jk}, \quad \forall (i,j) \in IP \tag{11}$$

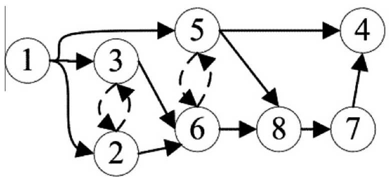

Fig. 1. Precedence relationships (solid line arrows) and sequence dependent time increments (dashed line arrows) for the PC example.

Table 1
Knowledge database for the PC example.

Part	Task i	Time t_i	Hazardous	Demand d_i
PC top cover	1	14	No	360
Floppy drive	2	10	No	500
Hard drive	3	12	No	620
Back plane	4	18	No	480
PCI cards	5	23	No	540
RAM modules	6	16	No	750
Power supply	7	20	No	295
Motherboard	8	36	No	720

The first objective given in Eq. (4) is to minimize the number of workstations for a given cycle time (the maximum time available at each workstation) (Battaia & Dolgui 2013; Baybars 1986). It rewards the minimum number of workstations, but allows the unlimited variance in the idle times between workstations because no comparison is made between station times. Thus, it does not force to minimize the total idle time of workstations. The second objective given in Eq. (5) is to aggressively ensure that idle times at each workstation are similar, albeit at the expense of generating a non-linear objective function (McGovern & Gupta 2007). The method is computed based on the minimum number of workstations required as well as the sum of the square of the idle times for all the workstations. This penalizes solutions where, even though the number of workstations may be minimized, one or more have exorbitant amount of idle times when compared to the other workstations. It also provides for leveling the workload between different workstations on the disassembly line. Therefore, a resulting minimum performance value is the more desirable solution indicating both a minimum number of workstations and similar idle times across all workstations. As the third objective (see Eq. (6)), a hazard measure is developed to quantify each solution sequence's performance, with a lower calculated value being more desirable (McGovern & Gupta 2007). This measure is based on binary variables that indicate whether a part is considered to contain hazardous material or not (the binary variable is equal to 1 if the part is hazardous, else 0) and its position in the sequence. A given solution sequence hazard measure is defined as the sum of hazard binary flags multiplied by their position number in the solution sequence, thereby rewarding the removal of hazardous parts early in the part removal sequence. As the fourth objective (Eq. (7)), a demand measure was developed to quantify each solution sequence's performance, with a lower calculated value being more desirable (McGovern & Gupta 2007). This measure is based on positive integer values that indicate the quantity required of a given part after it is removed (or 0 if it is not desired) and its position in the sequence. A solution sequence demand measure is then defined as the sum of the demand value multiplied by the position of the part in the sequence, thereby rewarding the removal of high demand parts early in the part removal sequence.

The constraints given in Eq. (8) ensure that all tasks are assigned to at least and at most one workstation which represents the complete assignment of each task. This constraint is necessary because of the assumption made in the first paragraph of Section 3 (i.e., a task cannot be divided between workstations).

Eq. (9) guarantees that the number of workstations with a workload does not exceed the permitted number. Here t_i is the part removal time for the ith of n parts where the cycle time, c, is the maximum amount of time available to complete all tasks assigned to each workstation and m^* is the theoretical minimum number of workstations. Because of Eq. (8), it is obvious that the minimum number of workstations cannot exceed the number of parts. The station time is the total time needed to execute all the tasks assigned to a station. The constraint given in Eq. (10) ensures that the work content of a workstation does not exceed the cycle time.

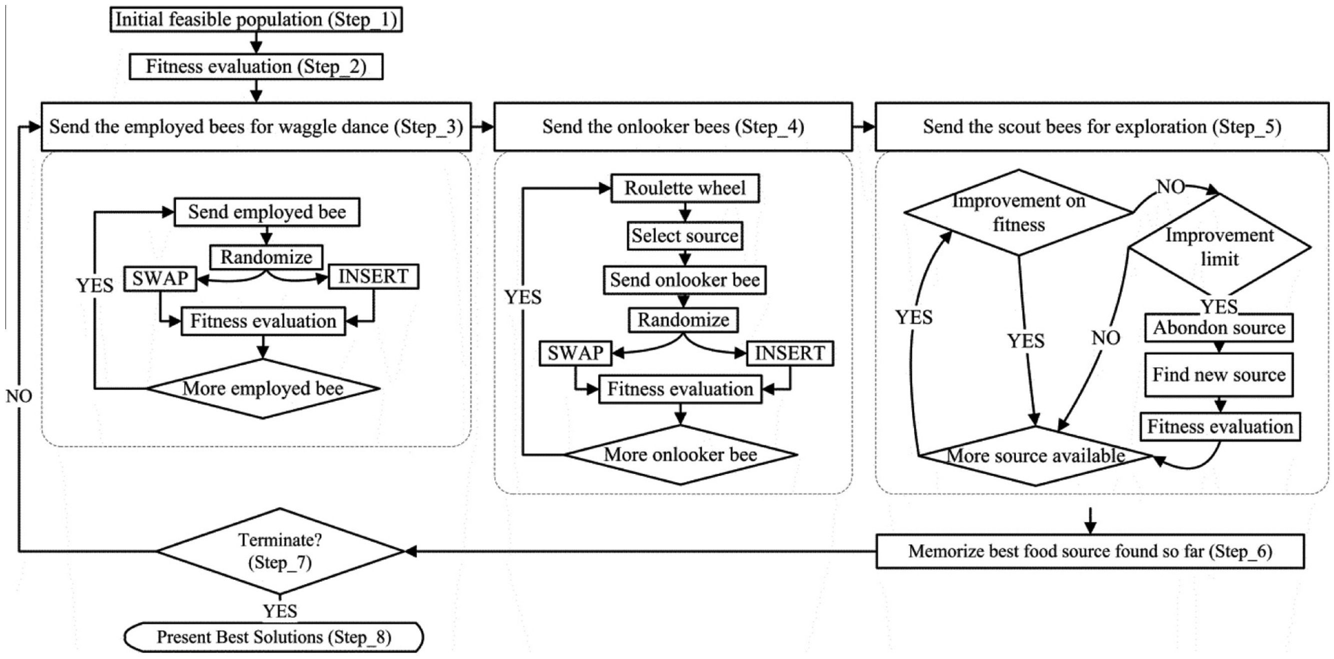

Fig. 2. Flowchart of the ABC algorithm.

Since the precedence relationships play a major role in the disassembly of a product, we need to ensure that the precedence relations among the tasks are not violated. The constraints given in Eq. (11) are designed to do just that. This constraint set allows the assignment of task i to station k if and only if all of its predecessors are already assigned somewhere between stations 1 through k, thus making sure that the precedence constraints are preserved.

4. Proposed artificial bee colony (ABC) approach

The ABC algorithm is a swarm based metaheuristic approach proposed in (Karaboga, 2005) to find optimal solutions in numerical optimization problems. This approach is inspired by the intelligent foraging behavior of honeybee swarm. There are few control parameters in the ABC algorithm, which is the main advantage of the algorithm. Due to its simplicity and ease of implementation, the ABC algorithm has recently gained more attention and has been used to solve many practical engineering problems.

The ABC algorithm classifies the foraging artificial bees into three groups, namely, employed bees, onlookers and scouts. A bee that is currently exploiting a food source is called an employed bee. The employed bees bring loads of nectar from the food source to the hive and may share the information about food source with onlooker bees. A bee waiting in the hive for making decision to choose a food source is named as an onlooker. A bee carrying out a random search for a new food source is called a scout. Employed bees share information about food sources by dancing in a common area in the hive called dance area. The duration of a dance is proportional to the nectar content of the food source currently being exploited by the dancing bee. Onlooker bees which watch numerous dances before choosing a food source tend to choose a food source according to the probability proportional to the quality of that food source. Therefore, the good food sources attract more bees than the bad ones. Scout bees can be visualized as performing the job of exploration, whereas employed and onlooker bees can be visualized as performing the job of exploitation.

The basic ABC algorithm was originally designed for continuous function optimization. In order to make it applicable for solving the SDDLBP with multiple objectives, a discrete variant of the ABC algorithm, is proposed. Similar to the other population based algorithms, ABC is an iterative process. The flow chart of the proposed ABC algorithm is given in Fig. 2.

The algorithm starts with scout bees being placed randomly in the search space. The fitness of the food sources visited by the scout bees are evaluated in step 2. In step 3, employed bees are sent onto the food sources for waggle dance. In step 4, onlooker bees are sent onto their food sources depending on their nectar amounts. In step 5, scout bees are sent to search for possible new food sources. The best food source found so far is memorized in step 6. Termination criterion is checked in step 7 and finally best solutions are presented in step 8. Pseudo code of the modified ABC for SDDLBP is given in Table 2.

4.1. Solution representation

A configuration is a solution to a given problem. In the proposed ABC algorithm, permutation based representation is used, so elements of a solution string are integers. Each element represents a task assignment to workstation. For example, if there are 8 tasks to be assigned to workstations then the length of the solution

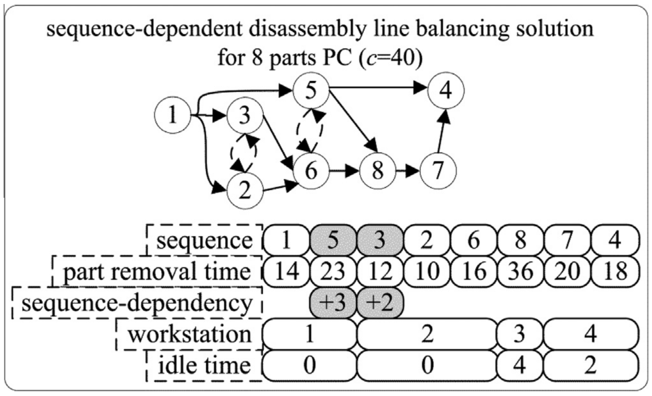
sequence-dependent disassembly line balancing solution for 8 parts PC (c=40)
1
3
5
4
2
6
8
7
sequence 1 5 3 2 6 8 7 4
part removal time 14 23 12 10 16 36 20 18
sequence-dependency +3 +2
workstation 1 2 3 4
idle time 0 0 4 2

Fig. 3. Assignment of tasks to workstations.

Table 2
Pseudo code of the modified ABC for SDDLBP.

```
Parameter Initialization
Send scout bees randomly for exploration
Evaluate scout bees' fitness functions $F_{sb} = [f_1, f_2, f_3, f_4]$
While $time < tlimit$
  For each employed bee
    Send employed bee to the next food source
    Apply one of neighborhood structures (SWAP or INSERT operators with .5 probability) to employed bee assigned to food source
    Evaluate employed bee's fitness functions $F_{eb} = [f_1, f_2, f_3, f_4]$
    Update best solution ($F_{best}$)
    Update rest iteration count as $ric = ric + 1$ if no improvement detected
  End for
  Calculate selection probability for each source considering $f_2$ since it has the dominant priority within other objective functions
  $PE_b = f_b / \sum_{b_1=1}^{eb} f_{b_1}$
  For each onlooker bee
    Select food source with $r \sim (0, 1)$ within cumulative $PE$ (selection probabilities vector for employed bees)
    Send onlooker bee to the selected food source
    Apply one of neighborhood structures (SWAP or INSERT operators with .5 probability)
    to employed bee assigned to food source
    Evaluate onlooker bee' fitness functions $F_{ob} = [f_1, f_2, f_3, f_4]$
    Update best solution ($F_{best}$)
    Update rest iteration count ($ric = ric + 1$) if no improvement detected
  End for
  For each scout bee
  If $ric \geqslant ilimit$
      Call scout bees and create random feasible line balance
      Evaluate scout bee's fitness functions
      Update best solution ($F_{best}$)
      Reset rest iteration count ($ric = 0$)
    End if
  End for
  Memorize best solution ($F_{best}$)
End While
```

string is 8. See Fig. 3 for an example of assignment of tasks to workstations.

4.2. Population initialization

Initial solutions are randomly generated for ABC such that each solution is assured to be feasible which forces the ABC search in the feasible solution space. Heuristics that are used to feed ABC by an initial feasible solution include Greatest Ranked Positional Weight, Longest Processing Time, Shortest Processing Time, Greatest Number of Immediate Successors, Greatest Number Of Successors, Random Priority, Smallest Task Number, Greatest Average Ranked Positional Weight, Smallest Upper Bound, Smallest (Upper Bound Divided by the Number of Successors), Greatest (Processing Time Divided by the Upper Bound), Smallest Lower Bound, Minimum Slack, Minimum (Number of Successors Divided by Task Slack), Greatest Number of Immediate Predecessors (Baykasoglu, 2006). A repair function is applied to ensure that each created solution is feasible.

The strategy of building a feasible balancing solution is the key issue in solving the SDDLBP. We use station-oriented procedure for a solution constructing strategy in which solutions are generated by filling workstations successively one after the other (Ding et al., 2010). The procedure is initiated by the opening of a first station. Then, tasks are successively assigned to this station until more tasks cannot be assigned and a new station is opened. In each iteration a task is randomly chosen from the set of candidate tasks to assign to the current station. When no more tasks may be assigned to the open station, this is closed and the following station is opened. The procedure finalizes when there are no more tasks left to assign. In order to describe the process to build a feasible balancing solution, available task and candidate task are defined as follows: A task is an *available task* if and only if it has not already been assigned to a workstation and all of its predecessors have already been assigned to a workstation. A task is a *candidate task* if and only if it belongs to the set of available task and the idle time of current workstation is higher than or equal to the processing time of the task. The generation procedure of a feasible balancing solution is given as follows:

Step 1: Start.
Step 2: According to the precedence constraints construct the available task set.
Step 3: According to the cycle time construct the candidate task set.
Step 4: If the set of candidate task is null, go to step 6.
Step 5: Select the task with the highest assignment priority from the candidate task set and assign the task to the current workstation; go back to step 2.
Step 6: If the set of available task is null, go to step 8.
Step 7: Open a new workstation, go back to step 2.
Step 8: Stop the procedure.

4.3. Employed bee phase

According to the basic ABC algorithm, the employed bees generate food sources in the neighborhood of their current positions. As to the permutation based neighborhood structure, INSERT (inserting a task to a different work station) and SWAP (interchanging two tasks) operators are used to produce neighboring solutions such that the new neighboring solutions are ensured to be feasible. By guaranteeing feasibility in each operation, the necessity of the repair function is prevented. The INSERT operator of a permutation is defined by removing a job from its original position and inserting it into another position whereas the SWAP operator produces a neighbor by interchanging two randomly selected tasks in the different positions at workstations while satisfying the precedence constraints. To enrich the neighborhood structure and diversify

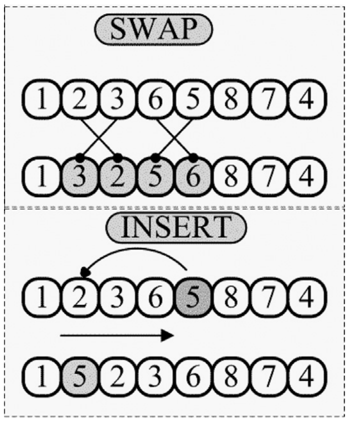

Fig. 4. SWAP and INSERT Operators.

the population, four neighboring approaches, based on the INSERT or SWAP operator, are separately utilized to generate neighboring food sources for the employed bees. Examples for SWAP and INSERT operators are given in Fig. 4.

4.4. Onlooker bee phase

In ABC, the onlookers utilize the global search operator to perform exploration search. An onlooker bee selects a food source depending on its winning probability value. After all employed bees complete their searches, they share their information related to the nectar amounts and the positions of their sources with the onlooker bees on the dance area. An onlooker bee evaluates the nectar information taken from all employed bees and chooses a food source site with a probability related to its nectar amount. This probabilistic selection depends on the fitness values of the solutions in the population. A fitness-based selection scheme might be a roulette wheel, ranking based, stochastic universal sampling, tournament selection or another selection scheme. In basic ABC, roulette wheel selection scheme in which each slice is proportional in size to the fitness value is employed as given below:

$$PE_b = f_b \bigg/ \sum_{b_1=1}^{eb} f_{b_1} \tag{12}$$

where f_b is the fitness value of solution carried by the bth employed bee. The higher the f_b value, the higher is the probability that the solution carried by the bth bee is selected. Although this is a multi-criteria problem, only a single criterion (the measure of balance, f_2) is being used in the ABC calculations. The other objectives are only considered after balance and then only at the completion of each cycle, not as part of the probabilistic node selection. That is, a bee's tour solution is produced based on f_2 (dominates f_1) while at the end of each cycle the best overall solution is then updated based on f_1, f_2, f_3, f_4 in that order. This is done because for the purpose of this study, the balance is considered by the overriding requirement, as well as to be consistent with previous multi-criteria DLBP studies (McGovern & Gupta, 2006).

4.5. Scout bee phase

A scout bee performs random search in the ABC algorithm. If a food source cannot be further improved through a predetermined number of trials limit, the food source is assumed to be abandoned, and the corresponding employed bee becomes a scout. The scout produces a food source randomly.

Table 3
Knowledge database for the 10-part product.

Task i	Time t_i	Hazardous	Demand d_i
1	14	No	0
2	10	No	500
3	12	No	0
4	17	No	0
5	23	No	0
6	14	No	750
7	19	Yes	295
8	36	No	0
9	14	No	360
10	10	No	0

5. Numerical results

The proposed algorithm was coded in both MATLAB and C++ and tested on Intel Core2 1.79 GHz processor with 3 GB RAM. After engineering, the program was investigated on two different scenarios. A full factorial set of experiments was performed to find the best *ilimit* parameter value. The best value was found to be 5 (*ilimit* = 5). The performance of the proposed ABC algorithm was also compared with that of ant colony optimization (ACO) metaheuristics, hybrid genetic algorithm (GA), particle swarm optimization (PSO), river formation dynamics (RFD) simulated annealing (SA) and tabu search (TS). All algorithms were executed 30 times (sample size) each in order to have sufficient statistical data for comparison since each algorithm had probabilistic and randomized features. *Two scenarios* were considered.

Scenario 1: The first scenario was for a product consisting of $n = 10$ components. The knowledge database and precedence relationships for the components are given in Table 3 and Fig. 5, respectively. The problem and its data were modified from McGovern and Gupta (2006) and consisted of a paced disassembly line operating at a speed which allows $c = 40$ s for each workstation to perform its required disassembly tasks. The sequence dependencies for the 10 part product are given as follows: $sd_{14} = 1$, $sd_{23} = 2$, $sd_{32} = 3$, $sd_{41} = 4$, $sd_{45} = 4$, $sd_{54} = 2$, $sd_{56} = 2$, $sd_{65} = 4$, $sd_{69} = 3$, $sd_{96} = 1$.

Scenario 2: The second scenario was a cellular telephone instance with $n = 25$ components. Gupta, Erbis, and McGovern (2004) provided the basis for this 25-part cellular telephone SDDLBP instance. McGovern and Gupta (2006) modified this instance from its original use in disassembly sequencing and augmented it with disassembly line specific attributes. Kalayci and Gupta (2013) added sequence-dependent part removal time

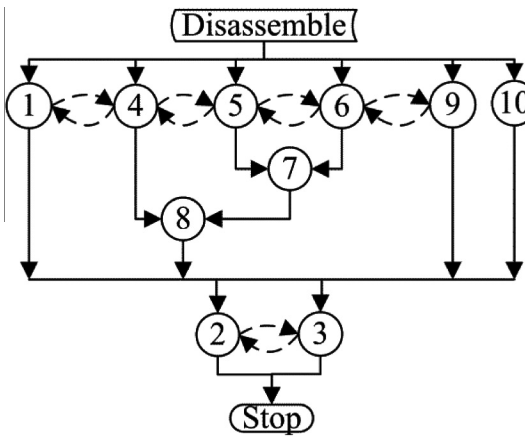

Fig. 5. Precedence relationships (solid line arrows) and sequence dependent time increments (dashed line arrows) for the 10 part product.

Table 4
Knowledge database of the cellular telephone instance.

Part	Task i	Part removal time t_i	Hazardous	Demand d_i
Antenna	1	3	Yes	4
Battery	2	2	Yes	7
Antenna guide	3	3	No	1
Bolt (Type 1) A	4	10	No	1
Bolt (Type1) B	5	10	No	1
Bolt (Type2) 1	6	15	No	1
Bolt (Type2) 2	7	15	No	1
Bolt (Type2) 3	8	15	No	1
Bolt (Type2) 4	9	15	No	1
Clip	10	2	No	2
Rubber Seal	11	2	No	1
Speaker	12	2	Yes	4
White cable	13	2	No	1
Red/Blue cable	14	2	No	1
Orange cable	15	2	No	1
Metal top	16	2	No	1
Front cover	17	2	No	2
Back cover	18	3	No	2
Circuit board	19	18	Yes	8
Plastic Screen	20	5	No	1
keyboard	21	1	No	4
LCD	22	5	No	6
Sub-keyboard	23	15	Yes	7
Internal IC board	24	2	No	1
Microphone	25	2	Yes	4

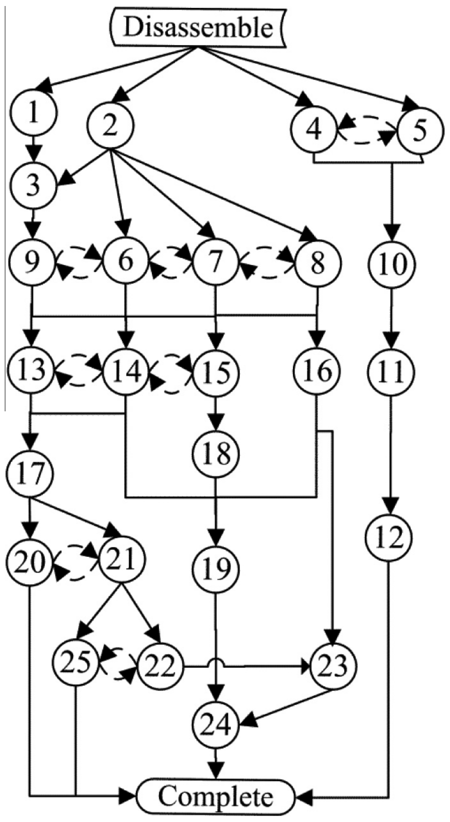

Fig. 6. Cellular telephone instance precedence relationships (solid line arrows) and sequence-dependent time increments (dashed line arrows).

influence on parts interacting with each other. This real-world instance consisting of $n = 25$ components has several precedence relationships as well as sequence-dependent obstructing parts. The data set includes a paced disassembly line operating at a speed which allows $c = 18$ s per workstation. In Table 4, the basic data including part removal time, information if the part is hazardous

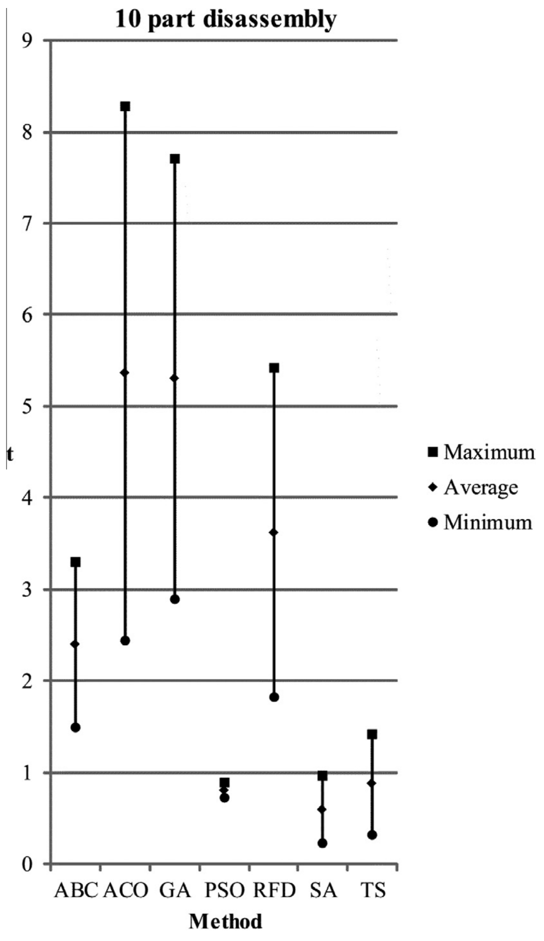

Fig. 7. Time performance comparison of ABC, ACO, GA, PSO, RFD, SA and TS for 10 parts disassembly.

or not, and demand of each part is given. Fig. 6 represents precedence relationships (solid line arrows) and sequence-dependent time increments (dashed line arrows) for the 25-part mobile phone disassembly instance. The sequence dependencies for this instance are as follows: $sd_{45} = 2$, $sd_{54} = 1$, $sd_{67} = 1$, $sd_{69} = 2$, $sd_{76} = 2$, $sd_{78} = 1$, $sd_{87} = 2$, $sd_{96} = 1$, $sd_{13\text{-}14} = 1$, $sd_{14\text{-}13} = 2$, $sd_{14\text{-}15} = 2$, $sd_{15\text{-}14} = 1$, $sd_{20\text{-}21} = 1$, $sd_{21\text{-}20} = 2$, $sd_{22\text{-}25} = 1$, $sd_{25\text{-}22} = 2$.

Results for Scenario 1: The solutions for the first scenario were found using all seven algorithms as well as the exhaustive search method. While, on average, the exhaustive search method was able to find optimal solution in 215t time, the proposed approach, ABC, on average, was able to successfully find the optimal solution in just over 2t time under the restriction of the system specifications given above. Note that the search space could be as large as 10! = 3,628,800. Table 5 depicts an optimal solution sequence. The fitness function values of the optimal solution were found to be: $f_1 = 5$, $f_2 = 67$, $f_3 = 5$, $f_4 = 9605$. According to this sequence, sequence dependent time increments sd_{56}, sd_{96}, sd_{41}, sd_{45}, sd_{32} are added to the part removal times of part 6, 6, 1, 5, 2 respectively.

Detailed average, standard deviation, standard error and confidence interval values for each objective are given in Tables 6 and 7. According to the numerical results for the first scenario, all of the methods provided were able to reach optimal solutions (which are, of course, guaranteed by exhaustive search method). From Table 6 it is clear that ABC, ACO, GA, PSO, RFD, SA and TS methods were able to find the optimal solutions for the first scenario in 2.40 t, 5.36 t, 5.30 t, 0.81 t, 3.62 t, 0.59 t and 0.87 t time, respectively. Here, ACO seemed to have the worst performance. Yet, within the 90% confidence interval, there was no statistically

Table 5
An optimal solution sequence for 10-part product disassembly.

		Workstations					
		I	II	III	IV	V	
Part removal sequence →	6	14(+2 + 1)					Time to remove parts (in seconds)
	1	14(+4)					
	10		10				
	5		23(+4)				
	7			19			
	4			17			
	8				36		
	9					14	
	2					10(+3)	
	3					12	
Total time		35	37	36	36	39	
Idle time		5	3	4	4	1	

Table 6
Average and standard deviation values for objectives.

Objective	Method	Scenario 1		Scenario 2	
		Average	Standard deviation	Average	Standard deviation
f_1	ABC	5	0.00	10	0.00
	ACO	5	0.00	10	0.00
	GA	5	0.00	10	0.00
	PSO	5	0.00	10	0.00
	RFD	5	0.00	10	0.00
	SA	5	0.00	10	0.00
	TS	5	0.00	10	0.00
f_2	ABC	67.00	0.00	10.07	2.16
	ACO	67.00	0.00	17.77	1.41
	GA	67.00	0.00	12.13	2.56
	PSO	67.00	0.00	13.97	1.96
	RFD	67.00	0.00	16.00	0.00
	SA	67.00	0.00	11.70	1.82
	TS	67.00	0.00	13.30	1.70
f_3	ABC	5.00	0.00	80.00	1.14
	ACO	5.00	0.00	82.80	1.32
	GA	5.00	0.00	79.77	0.73
	PSO	5.00	0.00	80.63	2.46
	RFD	5.00	0.00	80.60	0.62
	SA	5.00	0.00	83.43	3.22
	TS	5.00	0.00	83.10	2.87
f_4	ABC	9605.00	0.00	925.57	5.07
	ACO	9605.00	0.00	949.37	6.31
	GA	9605.00	0.00	924.90	2.40
	PSO	9605.00	0.00	932.50	11.53
	RFD	9605.00	0.00	939.83	2.29
	SA	9605.00	0.00	940.93	12.40
	TS	9605.00	0.00	941.30	12.51
t	ABC	2.40	1.73	124.60	146.79
	ACO	5.36	5.61	244.67	161.49
	GA	5.30	4.62	156.37	138.27
	PSO	0.81	0.16	40.74	24.71
	RFD	3.62	3.45	222.25	119.24
	SA	0.59	0.71	297.91	141.19
	TS	0.87	1.05	273.02	142.14

significant difference between ACO, ABC, GA and RFD. While SA appeared to perform the best, the difference between SA, PSO and TS was statistically insignificant. PSO seemed to be more consistent than the other algorithms in terms of speed while ACO, GA and RFD provided a larger time period interval. Thus, in terms of time performance for reaching optimal solutions faster, the algorithms are ranked in the following order from best to worst: SA = PSO = TS > ABC = RFD = GA = ACO. From this, ABC seems to have a reasonable performance when compared to the other algorithms in terms of speed. In fact, all algorithms reached optimal solutions very fast (in less than 9 t part removal time for the worst case) when compared to exhaustive search (which was logged at 215 t, on average). Results are demonstrated in Fig. 7.

Results for scenario 2: Since within the vast search space (25!), the exhaustive search is prohibitive, the optimal solution is unknown. Therefore, the best solutions found by the algorithms should be accepted as near optimal solutions. According to our estimates, exhaustive search method would require more than a year's time to solve the problem even if we use a super computer and change the platform to a faster programming language to solve the problem to optimality. Therefore, near optimal solutions found in reasonable computation times should satisfy our needs. The proposed ABC algorithm and the other approaches were able to find

Table 7
Standard error and confidence interval values for objectives.

Objective	Method	Scenario 1			Scenario 2		
		Standard error	90% Confidence interval		Standard error	90% Confidence interval	
f_1	ABC	0.00	5	5	0.00	10	10
	ACO	0.00	5	5	0.00	10	10
	GA	0.00	5	5	0.00	10	10
	PSO	0.00	5	5	0.00	10	10
	RFD	0.00	5	5	0.00	10	10
	SA	0.00	5	5	0.00	10	10
	TS	0.00	5	5	0.00	10	10
f_2	ABC	0.00	67.00	67.00	−0.71	9.35	10.78
	ACO	0.00	67.00	67.00	−0.46	17.30	18.23
	GA	0.00	67.00	67.00	−0.84	11.29	12.97
	PSO	0.00	67.00	67.00	−0.64	13.32	14.61
	RFD	0.00	67.00	67.00	0.00	16.00	16.00
	SA	0.00	67.00	67.00	−0.60	11.10	12.30
	TS	0.00	67.00	67.00	−0.56	12.74	13.86
f_3	ABC	0.00	5.00	5.00	−0.38	79.62	80.38
	ACO	0.00	5.00	5.00	−0.44	82.36	83.24
	GA	0.00	5.00	5.00	−0.24	79.53	80.01
	PSO	0.00	5.00	5.00	−0.81	79.83	81.44
	RFD	0.00	5.00	5.00	−0.20	80.40	80.80
	SA	0.00	5.00	5.00	−1.06	82.37	84.49
	TS	0.00	5.00	5.00	−0.94	82.16	84.04
f_4	ABC	0.00	9605.00	9605.00	−1.67	923.90	927.23
	ACO	0.00	9605.00	9605.00	−2.07	947.29	951.44
	GA	0.00	9605.00	9605.00	−0.79	924.11	925.69
	PSO	0.00	9605.00	9605.00	−3.79	928.71	936.29
	RFD	0.00	9605.00	9605.00	−0.75	939.08	940.59
	SA	0.00	9605.00	9605.00	−4.08	936.85	945.01
	TS	0.00	9605.00	9605.00	−4.12	937.18	945.42
t	ABC	−0.90	1.50	3.30	−48.29	76.31	172.89
	ACO	−2.92	2.44	8.27	−53.12	191.55	297.80
	GA	−2.41	2.90	7.71	−45.49	110.88	201.86
	PSO	−0.08	0.72	0.89	−8.13	32.61	48.87
	RFD	−1.79	1.83	5.41	−39.23	183.02	261.47
	SA	−0.37	0.22	0.96	−46.45	251.46	344.36
	TS	−0.55	0.33	1.42	−46.76	226.26	319.78

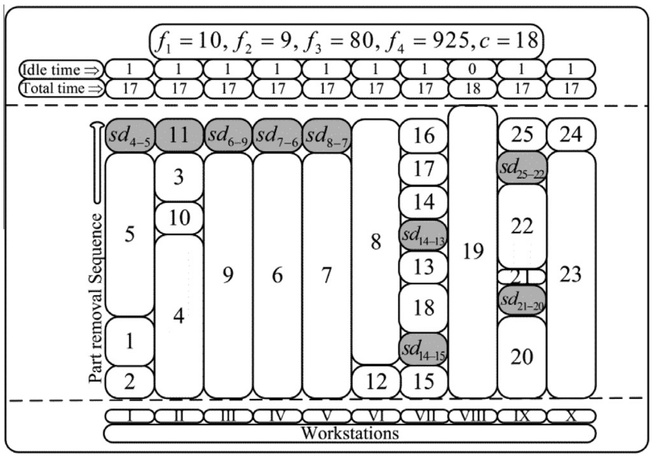
$f_1 = 10, f_2 = 9, f_3 = 80, f_4 = 925, c = 18$
Idle time
Total time
Part removal Sequence
Workstations
I II III IV V VI VII VIII IX X

Fig. 8. A typical solution found using the cellular telephone instance.

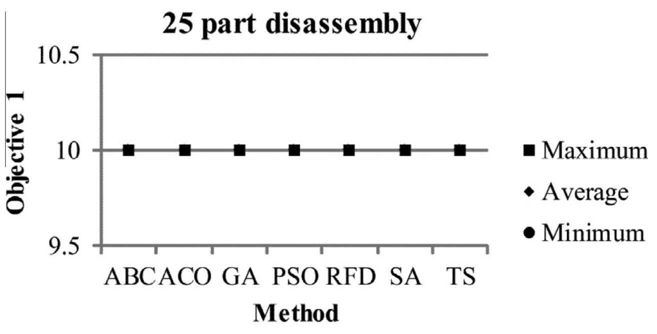

Fig. 9. f_1 Performance comparison of ABC, ACO, GA, PSO, RFD, SA and TS for 25 parts cellular telephone instance.

the best known solution given in Fig. 8 within a reasonable time of 350t.

Table 6 shows the average and standard deviation values for the numerical results of each algorithm for the second scenario. For example, in Table 6, one can see that all approaches found the minimum number of workstations as 10 (f_1 = 10) (see Fig. 9). Since f_1 has the highest priority among all objectives and all algorithms were able to find the same objective value, we next need to check the performance of the second objective (f_2) for a comparison.

Table 6 shows that ABC, ACO, GA, PSO, RFD, SA and TS found 10.07, 17.77, 12.13, 13.97, 16.00, 11.70, 13.30 respectively as average values for the second objective (f_2), and Table 7 shows the standard errors as −0.71, −0.46, −0.84, −0.64, 0.00, −0.60, −0.56, respectively. Within 90% confidence interval, ABC algorithm statistically outperformed the other approaches provided. Fig. 10 also depicts that ABC algorithm had the best performance. Therefore ABC was ranked as the best algorithm. As can also be seen in Fig. 10, ACO had the worst performance among all algorithms in terms of the second objective (f_2) value. In addition, RFD was ranked as the second worst algorithm. While TS performed slightly better than PSO, the difference between them was not statistically significant within 90% confidence interval. GA and SA were better than PSO and there was not a significant difference between GA and SA. There was not a significant difference between GA and TS, either. However, SA was better than TS. We need to further investigate f_3 next.

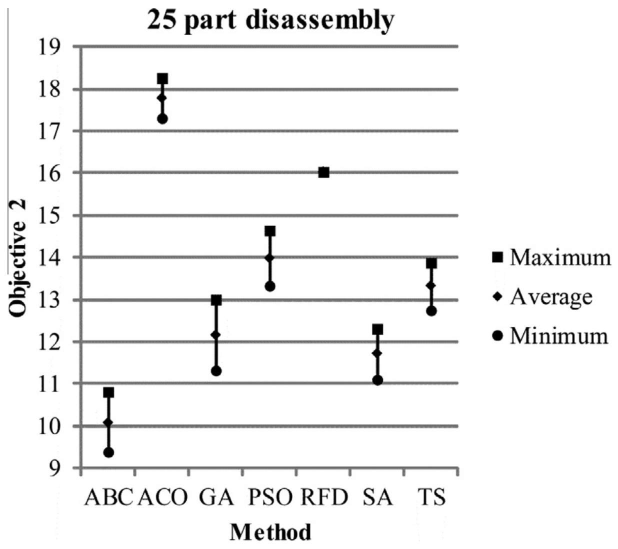

Fig. 10. f_2 Performance comparison of ABC, ACO, GA, PSO, RFD, SA and TS for 25 parts cellular telephone instance.

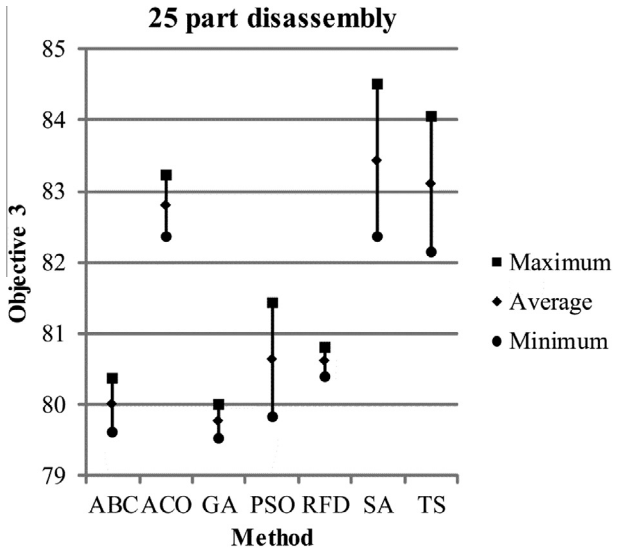

Fig. 11. f_3 Performance comparison of ABC, ACO, GA, PSO, RFD, SA and TS for 25 parts cellular telephone instance.

It is clear from Fig. 11 that GA and PSO were better than TS in terms of the third objective (f_3) performance. Therefore, TS algorithm was ranked as third worst algorithm among them. Thus (remembering from Fig. 10), since GA and SA was better than PSO, PSO algorithm was ranked as the fourth worst algorithm. In Fig. 11, one can see that GA was better than SA. Therefore, GA was ranked as second best while SA was ranked as third best algorithm. Since the fourth objective (f_4), has the least priority, it is not required to check f_4 performance for comparison. The results are provided (see Fig. 12) for general information purposes only. Based on the above discussion, the algorithms were ranked from best to worst in the following order: ABC > GA > SA > PSO > TS > RFD > ACO.

In this section, the results of seven combinatorial optimization methods applied to the Sequence-Dependent Disassembly Line Balancing Problem (SDDLBP) were quantitatively and qualitatively compared. All of the methodologies performed extremely well when compared to exhaustive search. While different techniques showed varying degrees of performance strength, the proposed

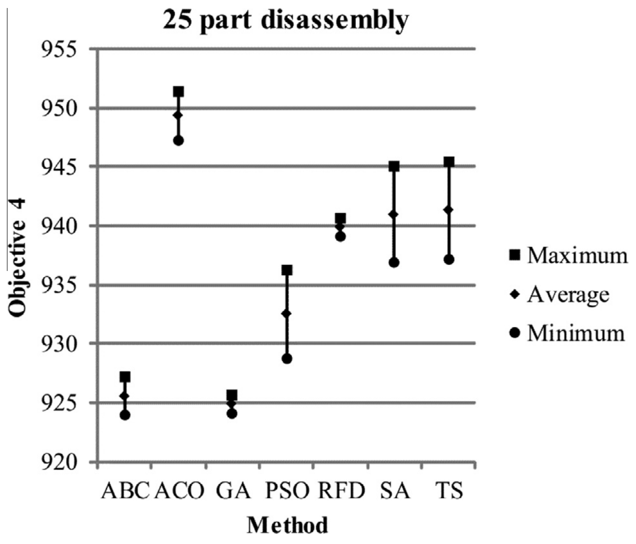

Fig. 12. f_4 Performance comparison of ABC, ACO, GA, PSO, RFD, SA and TS for 25 parts cellular telephone instance.

ABC had an overall superior performance compared to others even though it was slightly slower than SA, TS and PSO. Since the proposed algorithm was specifically designed to solve the SDDLBP, the results only apply to this type of problems and do not comment on any one technique to be best for all applications.

6. Conclusions

Efficient product recovery has become increasingly significant. As the first critical and most time consuming step of product recovery, disassembly faces many unique challenges. Sequence-dependent disassembly line balancing problem considered in this paper is a recently reported multi-objective NP-complete optimization problem. The main objective of this paper was to solve the SDDLBP. An ABC algorithm was presented to solve two primary problem instances, one small sized and the other large sized, and was compared to six very fast, near (optimal) combinatorial optimization approaches. Although exhaustive search consistently provides an optimal solution, its exponential time complexity quickly reduces its practicality. Heuristic methodologies are efficient in obtaining near optimal solutions to problems with intractably large solution spaces. The proposed method sought to provide a feasible disassembly sequence, minimize the number of workstations, minimize the total idle time and minimize the variation of idle times between workstations while attempting to remove hazardous and high-demand components as early as possible in a disassembly line. It is well suited to the multi-criteria decision making problem format as well as for the solution of problems with nonlinear objectives. In addition, it is ideally suited to integer problems which are not easily solved by traditional optimum solution generating mathematical programming techniques. The conclusions drawn from the study include the consistent generation of (near) optimal solutions, the ability to preserve precedence relationships, the superior performance of the proposed method, and its practicality due to the ease of implementation in solving sequence-dependent disassembly line balancing problem.

As future research directions, combinatorial optimization methodologies provided may be further investigated and improved in terms of time complexity and objective performances. Other types of combinatorial optimization methodologies might show interesting results when applied to SDDLBP. Combinatorial optimization algorithms provided can be combined to work together in some manner to improve the solution and time complexity performance. The three problem instances (viz., 8, 10 and 25 parts instances) provide a variety of data sets differing in size, number, type of precedence constraints and sequence dependent time increments, however, consideration should be given to developing other data sets including those from actual problems as well as specifically designed experimental benchmarks. All of the part removal times in this study were deterministic; actual part removal times on a disassembly line are more accurately represented probabilistically, so a study making use of this type of data is warranted and development of data sets with probabilistic part removal times is necessary. Throughout the research, complete disassembly was assumed since it allows for a worst case study in terms of problem complexity and it provides a consistency across the problem instances and methodologies; in practical applications it is not necessarily desired, required, practical or efficient which recommends that future studies may consider more applied problem that allows for incomplete or partial disassembly. In addition to the terms given above, any further developments and applications of recent methodologies to the Sequence-Dependent Disassembly Line Balancing Problem will help to extend the focus area of the research.

References

Agrawal, S., & Tiwari, M. K. (2008). A collaborative ant colony algorithm to stochastic mixed-model U-shaped disassembly line balancing and sequencing problem. *International Journal of Production Research, 46*, 1405–1429.

Altekin, F. T., & Akkan, C. (2012). Task-failure-driven rebalancing of disassembly lines. *International Journal of Production Research, 50*, 4955–4976.

Altekin, F. T., Kandiller, L., & Ozdemirel, N. E. (2008). Profit-oriented disassembly-line balancing. *International Journal of Production Research, 46*, 2675–2693.

Battaia, O., & Dolgui, A. (2013). A taxonomy of line balancing problems and their solution approaches. *International Journal of Production Economics, 142*, 259–277.

Baybars, I. (1986). A survey of exact algorithms for the simple assembly line balancing problem. *Management Science, 32*, 909–932.

Baykasoglu, A. (2006). Multi-rule multi-objective simulated annealing algorithm for straight and U type assembly line balancing problems. *Journal of Intelligent Manufacturing, 17*, 217–232.

Che, Z.-G., Che, Z. H., & Hsu, T. A. (2009). Cooperator selection and industry assignment in supply chain network with line balancing technology. *Expert Systems with Applications, 36*, 10381–10387.

Ding, L.-P., Feng, Y.-X., Tan, J.-R., & Gao, Y.-C. (2010). A new multi-objective ant colony algorithm for solving the disassembly line balancing problem. *The International Journal of Advanced Manufacturing Technology, 48*, 761–771.

Go, T. F., Wahab, D. A., Rahman, M. N. Ab., Ramli, R., & Hussain, A. (2012). Genetically optimised disassembly sequence for automotive component reuse. *Expert Systems with Applications, 39*, 5409–5417.

Gungor, A., & Gupta, S. M. (1999). Issues in environmentally conscious manufacturing and product recovery: A survey. *Computers & Industrial Engineering, 36*, 811–853.

Gungor, A., & Gupta, S. M. (2001). A solution approach to the disassembly line balancing problem in the presence of task failures. *International Journal of Production Research, 39*, 1427–1467.

Gungor, A., & Gupta, S. M. (2002). Disassembly line in product recovery. *International Journal of Production Research, 40*, 2569–2589.

Gupta, S. M. (Ed.). (2013). *Reverse Supply Chains: Issues and Analysis*. Boca Raton: CRC Press.

Gupta, S. M., Erbis, E., & McGovern, S. M. (2004). Disassembly sequencing problem: A case study of a cell phone. In S. M. Gupta (Ed.). *Environmentally Conscious Manufacturing IV* (Vol. 5583, pp. 43–52). Bellingham: SPIE-International Society for Optical Engineering.

Ilgin, M. A., & Gupta, S. M. (2010). Environmentally conscious manufacturing and product recovery (ECMPRO): A review of the state of the art. *Journal of Environmental Management, 91*, 563–591.

Ilgin, M. A., & Gupta, S. M. (2012). *Remanufacturing Modeling and Analysis*. CRC Press.

Kalayci, C. B., & Gupta, S. M. (2011a). A hybrid genetic algorithm approach for disassembly line balancing. In *Proceedings of the 42nd annual meeting of decision science institute (DSI 2011)* (Vol. 1, pp. 2142–2148). Boston, MA, USA.

Kalayci, C. B., & Gupta, S. M. (2011b). Tabu search for disassembly line balancing with multiple objectives. In *41st International conference on computers and industrial engineering (CIE41)* (pp. 477–482). University of Southern California, Los Angeles, USA.

Kalayci, C. B., & Gupta, S. M. (2013). Ant colony optimization for sequence-dependent disassembly line balancing problem. *Journal of Manufacturing Technology Management, 24*, 413–427.

Kalayci, C. B., & Gupta, S. M. (2012). A particle swarm optimization algorithm for solving disassembly line balancing problem. In *Proceedings of northeast decision sciences institute 2012 annual conference* (pp. 347–357). Newport, Rhode Island, USA.

Kalayci, C. B., Gupta, S. M., & Nakashima, K. (2011). Bees colony intelligence in solving disassembly line balancing problem. In *Proceedings of the 2011 Asian conference of management science and applications (ACMSA2011)* (pp. 34–41). Sanya, Hainan, China.

Kalayci, C. B., Gupta, S. M., & Nakashima, K. (2012). A simulated annealing algorithm for balancing a disassembly line. In M. Matsumoto, Y. Umeda, K. Masui, & S. Fukushige (Eds.), *Design for innovative value towards a sustainable society* (pp. 714–719). Netherlands: Springer.

Karaboga, D. (2005). An idea based on honey bee swarm for numerical optimization. In Technical report TR06. Kayseri, Turkey: Computer Engineering Department, Engineering Faculty, Erciyes University..

Koc, A., Sabuncuoglu, I., & Erel, E. (2009). Two exact formulations for disassembly line balancing problems with task precedence diagram construction using an AND/OR graph. *IIE Transactions, 41*, 866–881.

McGovern, S. M., & Gupta, S. M. (2006). Ant colony optimization for disassembly sequencing with multiple objectives. *The International Journal of Advanced Manufacturing Technology, 30*, 481–496.

McGovern, S. M., & Gupta, S. M. (2007). A balancing method and genetic algorithm for disassembly line balancing. *European Journal of Operational Research, 179*, 692–708.

McGovern, S. M., & Gupta, S. M. (2011). *The Disassembly Line: Balancing and Modeling*. New York: McGraw Hill.

Scholl, A., Boysen, N., & Fliedner, M. (2006). The sequence-dependent assembly line balancing problem. *OR Spectrum, 30*, 579–609.

Tovey, C. A. (2002). Tutorial on Computational Complexity. *Interfaces, 32*, 30–61.

Wang, H.-F., & Gupta, S. M. (2011). *Green supply chain management: Product life cycle approach*. New York: McGraw Hill.

Wang, H. S., Che, Z. H., & Chiang, C. J. (2012). A hybrid genetic algorithm for multi-objective product plan selection problem with ASP and ALB. *Expert Systems with Applications, 39*, 5440–5450.