Design of Energy Efficient RAL System using Evolutionary Algorithms

1. Introduction

Assembly is considered to be one of the important processes in a manufacturing sector. Energy is one of the most important vital resources in a manufacturing scenario. Industries consume about half of the world's energy and the energy consumption has doubled over the years (Mouzon et al., 2007). Due to the serious environmental impacts and rise in the energy cost, there has been a growing interest for the development of energy efficient manufacturing systems (Dai et al., 2013).By the reduction of energy consumption, it helps the industries to save cost and become more competitive Manufacturing a car (Press, body, paint and assembly shops) consume energy about 700kwh/vehicle. This energy cost is about 9-12% of the total manufacturing cost and a 20% reduction in energy cost shall be about 2-2.4% reduction in the final manufacturing cost (Fysikopoulos et al., 2012). The economic importance of assembly as a manufacturing process has led to extensive efforts for improving the efficiency and cost effectiveness of assembly operations (Rubinovitz et al., 1993).

Assembly lines are considered to be used in the last step of production, where the final assembly of the product from previously made parts is carried out. In an assembly line, several workstations are available for performing a specific set of tasks and the product moves through the line, from one workstation to the other according to the order (Kumar, 2013). Assembly production lines can be manually operated, automated, or of mixed design. In an assembly line, tedious and repetitive tasks some of which are dangerous for workers are carried out (Rubinovitz et al., 1993).

Assembly Line Balancing (ALB) is the term commonly used to refer to the decision process of assigning tasks to workstations in a production system. Salveson first mathematically formulated the model for assembly line balancing in 1955 (Salveson, 1955). Assembly Line Balancing (ALB) problems mainly deals with assigning tasks to workstations in such a way that the assignment is in a balanced manner. Assembly line could be classified into two types based on the flow of tasks: straight assembly lines and U-shaped assembly line (Miltenburg and Wijngaard, 1994). In a straight-line layout, workers work on a continuous length of the line

where as in a U type layout, operators are allowed to work across both "sides" of the line. For U-shaped assembly lines entrance and exit are in the same position (Erel et al., 2001). In U-shaped assembly lines task can be assigned to a workstation after all its predecessor or all successors are assigned to an earlier or the same workstation. The distinguishing feature of U-shaped assembly line balancing problems allows for the forward and backward assignment of tasks to workstations (Baykasoğlu and Özbakır, 2007).

In the past decades, robots have been widely used in assembly systems as called robotic assembly lines. An assembly robot can work 24 hours a day without worries of fatigue. Productivity, quality of products and flexibility in manufacturing are few of the advantages of employing robots in an assembly line (Dai et al., 2013). Robots are available with different capabilities and efficiencies are available in the market. Hence, proper allocation of the robot is critical for the performance of the assembly line. The robotic assembly line balancing (RALB) problem aims at assigning tasks to the workstations and proper allocation of robot is required for improving the efficiency and productivity of the assembly line. Robotic assembly line balancing (RALB) problem is the generalized form of tradition assembly line balancing problem. RALB problems are divided on the basis of the objective. Two types of RALB problems are addressed in the literature. Type I RALB- Minimize the number of workstations when cycle time is fixed the objective is to and the objective of Type II RALB is to minimize the cycle time for given a fixed number of workstations. Rubinovitz and Bukchin (1991) first formulated type I RALB problem with an objective of minimizing the number of workstations for a fixed cycle time. Proposed model aims at allocating equal amounts of tasks to the work-stations while allocating the most efficient robot type from the given set of available types. Later, Rubinovitz et al. (1993) developed a branch and bound method to solve this problem.

Levitin et al. (2006) first developed an algorithm for solving type II RALB where the objective is to distribute tasks to the workstations in a balanced manner and allocate the best fit robot to perform these tasks with the objective of minimizing the cycle time when the number of workstations are fixed. Genetic algorithm is proposed to solve this problem. Two heuristics for assignment of tasks and evaluation of cycle time is also proposed. Gao et al. (2009) presented a 0-1 integer programming formulation for the RALB-II. A hybrid genetic algorithm is proposed to solve the problem. Partial representation technique is implemented for genetic algorithm. New

crossover and mutation operators to adapt to the chromosome structure and nature of the problem are proposed. Using a local search procedure the search ability of the algorithm is increased. Yoosefelahi et al. (2012) formulated a multi-objective mixed integer linear programming model for type-II RALB problem aims to minimize the cycle time, robot setup costs and robot costs. Daoud et al. (2014) proposed several evolutionary algorithms and a discrete event simulation model to solve robotic assembly line balancing problem and an automated packaging line dedicated for dairy food products is the case study considered for evaluating the proposed model. Table 1 summarizes the literature on RALB studies and are categorized according to the objectives and optimization techniques.

Table 1 Summary of research on RALB problems

Model	Objective	Procedure	Reference
Assignment of tasks to non-identical workstations without violating precedence relations.	Minimize the cost of workstations	Dynamic Programming	Nicosia et al. (2002)
RALB-1 problem	Minimize the number of workstations	-	Rubinovitz and Bukchin (1991)
RALB-1 problem	Minimize the number of workstations	Branch & Bound	Rubinovitz et al. (1993)
RALB problem	Minimize the equipment cost	An exact and heuristic branch & bound	Bukchin and Tzur (2000)
RALB-2 problem	Minimize the cycle time	Two versions of Genetic Algorithm	Levitin et al.(2006)
RALB-2 problem	Minimize the cycle time	Hybrid Genetic Algorithm	Gao et al.(2009)
RALB-2 problem	Minimize the cycle time, robot cost, setup costs	Three versions of multi-objective evolution strategies	Yoosefelahi et al.(2012)
RALB -E problem	Maximize line efficiency	Three evolutionary algorithms & Local Search	Daoud et al.(2014)

Literature on assembly line balancing problems shows that the key objectives evaluated are cost, time, and quality. However, the research on minimizing energy consumption in manufacturing systems has been rather limited (Dai et al., 2013). In case of RALB, most of the researchers considered only type-I and type-II robotic assembly line balancing problems for assigning tasks and allocating robots to workstations in a straight line assembly line. Literatures related to

assembly line balancing problems with the objective of minimizing energy consumption are limited and few of them are briefly discussed below.

Fysikopoulos et al. (2012) presented an empirical study of energy consumption in an automotive assembly line. They showed that by modeling an assembly line by including energy considerations, one can possibly save energy and cost. Luo et al. (2013) proposed a multi-objective ant colony optimization meta-heuristic to optimize electric power cost (EPC) with the presence of time-of-use (TOU) electricity prices. Dai et al. (2013) presented a mathematical model for a flexible flow shop schedulin (FFS) and proposed an energy-efficient model for FFS. Due to NP-hard nature of the problem, an improved genetic-simulated annealing algorithm is adopted. Shrouf et al. (2014) proposed a mathematical model to minimize the energy consumption cost for a single machine production system considering variable energy prices during a day. A genetic algorithm (GA) is proposed to obtain 'near' optimal solutions.

Since the simplest version of assembly line balancing problem falls under the category of NP-hard (Gutjahr and Nemhauser, 1964), researchers use optimization or simulation models to solve the problems. Researchers in the recent past have suggested that both exact methods and evolutionary algorithms can be used to solve assembly line balancing problems. Exact methods guarantee an optimum solution, whereas evolutionary algorithms attempt to yield a good solution. Time taken by an exact method to find an optimum solution for NP-hard problem will be much greater than the time taken by any heuristic method. Hence, heuristic methods are used for solving problems which are generally complex (Martí and Reinelt, 2011)

The literature survey reveals that till date no research has been done on U-shaped robotic assembly line balancing problem. This research aims at proposing an algorithm to solve U-shaped robotic assembly line problem by using evolutionary algorithms like Particle swarm optimization (PSO) and Differential Evolution (DE) algorithms. In this paper, an objective of obtaining maximum line efficiency for a fixed number of workstations for U-shaped assembly lines by minimizing the total energy consumption of the robotic U-shaped assembly line is considered. The remaining of this paper is organized as follows: Section 2 describes the problem formulation of RALB problem addressed in this paper along with assumptions considered. Section 3 details the implementation of PSO and DE. Section 4 presents the experimental results and discussions and it is concluded in Section 5.

2. Problem definition and Mathematical Model

This paper proposes an algorithm for a robotic U-shaped assembly line where different assembly tasks are to be performed by each workstation and produce a given product. A set of workstations and robots are considered in the assembly line. In a balanced assembly line, tasks needs to be assigned to the workstations and best robot needs to be allotted to the station to perform the assembly tasks. This algorithm minimizes the energy consumption of the robots at each workstation and tries to maximize the line efficiency of the U-shaped robotic assembly line.

The assumptions considered for the RALB problem are the following.

1. Robots power consumptions are assumed. Using the power values of each robot, energy consumption is calculated.
2. Model is designed for a U-shaped robotic assembly line.
3. Tasks cannot be subdivided among one or two work stations but any task can be assigned at any station and performed by any robot.
4. Time taken to perform a task depends on the robot assigned.
5. All type of robots is available without any limitations.
6. Material handling, loading and unloading time, as well as set-up and tool changing time are negligible, or are included in the task time.
7. The planning horizon is not included in the model. The maintenance operations are not considered in this study.

3. Evolutionary algorithms to solve RALB problem

Since the simplest form of assembly line balancing problem falls under the category of NP- hard many evolutionary algorithms have been developed to solve these problems since 1950's (Scholl and Becker, 2006). The problem address here is also NP-hard. Rashid et al. (2012) shows the different types of evolutionary algorithms applied on assembly line balancing problems. In this paper, two evolutionary algorithms are proposed to solve this problem: Particle Swarm Optimization and Differential Evolution. Following sections explains these algorithms in detail.

3.1 Particle Swarm Optimization

Particle swarm optimization (PSO) is a population-based, self-adaptive search optimization technique introduced by Kennedy and Eberhart (1995). The method was developed based on the animal social behaviors such as fish schooling, bird flocking, etc. PSO is used to solve engineering optimization problems due to the simple principle and ease of implementation (Kaveh and Talatahari, 2011). Particle Swarm optimization algorithm starts with a population of randomly generated population of individuals (particles) in the search space. PSO algorithm works on the social behavior of particles in the swarm. Each particle flies through the solution space and adjust its flying velocity and position using its own flying experience and the partners experience. Each particle updates its position and velocity according to the formula as follows (Ratnaweera et al., 2004):

$$P_i^{t+1} = P_i^t + v_i^{t+1} \tag{1}$$

$$v_i^{t+1} = c_1 v_i^t + c_2 U_1 ({}^{e}P_i^t - P_i^t) + c_3 U_2 (G^t - P_i^t) \tag{2}$$

Where U_1 and U_2 are the velocity coefficients (random numbers between 0 and 1), v_i^t is the initial velocity, ${}^{e}P^t_i$ is the Local best, G^t is the Global best and P_i^t is the current particle position, c_1 and c_2 and c_3 are the learning coefficients. ${}^{e}P^t_i$ refers to the best position of the i^{th} particle at generation 't'. G^t refers to the best position of all particles at generation 't'.

PSO method is becoming very popular due to its simplicity of implementation and ability to quickly converge to a reasonably good solution. Pseudo code of PSO is in Figure 1.

3.1.1 Population Initialization

The main step in the functioning of a PSO is the generation of an initial population. Each member (particle) of this population encodes a potential solution for the problem. The particle represents a sequence of numbers (tasks) arranged such a way that it meets the precedence relationship. Instead of starting the evolutionary algorithms with a set of random particles, a set of priority rules reported in the literature are used to generate a set of initial population. Six out of fourteen rules available in (Ponnambalam et al., 2000) are selected to create the set of initial population and remaining particles are randomly generated. Using the energy consumption data these particles are generated such a way precedence is not violated. In this research a total of 25

particles in the initial population are considered. Example of the sequences generated using these priority rules is presented in (Mukund Nilakantan and Ponnambalam, 2012). After the creation of initial population, each particle is evaluated and assigned a fitness value according to the fitness function. Section 3.1.3 explains how the fitness value is evaluated.

```
procedure PSO
  input (problem data, PSO parameters)
  begin
      t ← 0;
      for(i = 1, N)
          Generate  P_i^t;
              Fitness Evaluation Z (P_i^t);//Refer Section 3.1.3
          ^eP_i^t ← P_i^t;
      end
      G ← P_i^t having min {Z(^eP_i^t);  i = 1, N }
      for(i = 1, N)
          Initialize  v_i^t;
      end
      do {
              for (i = 1, N)
                      Update Position  P_i^{t+1} (Equation 1)
                      Update Velocity  v_i^{t+1}  (Equation 2 )
              end
                      Evaluate all particles //Refer Section 3.1.3
              Update ^eP_i^t and G, (i = 1, N)
              t ← t + 1
      } ( while (t < t_max))
      output G
  end;
```

Figure 1 Pseudo code of PSO

3.1.2 Velocity and Position Update

Various operations performed for computing particle velocity and updating particle positions (Rameshkumar et al., 2005) are explained below:

Subtraction (position – position) operator: Let us assume to positions x_1 and x_2 representing two different task sequences. The difference of x_2 - x_1 is a velocity v. In the Equation 2, for example subtracting two positions i.e. $({}^{e}p_i^t - P_i^t)$ results in a velocity which is a set of transpositions.

Addition (position + velocity) operator: Let us assume x to be the position and v to be the velocity of the particle. New position x_1 is calculated by applying the first transposition of v to x, i.e, $x_1 = x + v$ then the second one to the result etc.

Addition (velocity + velocity) operator: Let us assume two velocities v_1 and v_2. In order to calculate $v_1 + v_2$, the list of transpositions which contains first the 'ones' of v_1, followed by the 'ones' of v_2 is considered.

Multiplication (Coefficient × velocity) operator: Let c be the learning coefficient and v be the velocity. $c \times v$ results in a new velocity.

Using Equation 2, velocity is updated using the local best and global best for all iterations. Particle moves from their current position to new position using Equation 1. Each particle position is updated by adding the velocity vector to the current particle position in every generation.

An example is shown to explain how velocity and position updates works.

Local Best ${}^{e}P^{t}_{i}$ *: (1,2,6,3,4,5,7,8,10,9,11),Global Best G: (1,2,3,4,5,6,7,8,9,10,11),*

Particle P_i^t *: (1,2,3,6,5,4,7,8,10,9,11) and Initial velocity* v_i^t*: (2, 3) (4, 5).*

Learning coefficients: c_1=1, c_2=1 and c_3=2

U_1 and U_2 are generated randomly. Assuming U_1=0.8 and U_2 =0.3

Velocity of the particle is calculated using Equation 2.

v_i^{t+1}=(2,3)(4,5)+0.8x[(1,2,6,3,4,5,7,8,10,9,11)-(1,2,3,6,5,4,7,8,10,9,11)]+

0.6x [(1,2,3,4,5,6,7,8,9,10,11)-(1,2 ,3,6,5,4,7,8,10,9,11)]

=(2,3)(4,5)+0.8 x (2,3)(4,5) +0.6x(3,5)(8,9) = (2,3)(4,5)(8,9)

Using Equation 1 position of the particle is updated.

P_i^{t+1}= (1,2,3,6,5,4,7,8,10,9,11)+ (2,3) (4,5)(8,9) = (1,2,6,3,4,5,7,8,9,10,11)

In this research initial velocity of the particles are randomly generated and it is represented as number of transpositions. Depending on the size of the problem, the number of velocity pair varies. More the number of tasks, the number of velocity pairs are high. Table 2 shows the number of velocity pairs used in this research. From the table it could be understood that for small size problem within the range of 0-20, number of velocity pairs is 4. And when the velocity is updated in each iteration the number of velocity pair is restricted to maximum of 4. If excess pairs are formed it is discarded for the next iteration.

Table 2 Maximum Number of Velocity Pairs

Task Range	Maximum Velocity Pairs
0-20	4
20-40	8
40-60	10
60-80	25
80-100	30
120-140	50
140-200	65
200-300	75

3.1.3 Fitness Evaluation

A consecutive heuristic procedure proposed by Levitin et al. (2006) for straight line assembly line is adopted in this paper. This heuristic allocates tasks and robots to workstations with an objective of minimizing energy consumption of the assembly line. Procedure tries to allocate maximum number of tasks to be performed at each work station. For U-shaped assembly line tasks are to be allocated into the workstations by moving forward and backward through the precedence diagram in contrast to a typical forward move in the traditional assembly systems. Robots are checked for allotment to the workstation to perform the tasks allotted in it. Initial energy consumption E_0 of the assembly line is to be considered to start the procedure. Tasks are assigned to the workstations until the sum of energy consumption of the tasks is less than or equal to E_0. And the robot which performs these tasks at minimum energy consumption is allotted. If it is not possible to assign the tasks within the initial E_0, E_0 is incremented by one and procedure is repeated until all the tasks gets assigned. An illustration is provided in this section which explains the task and robot allocation. Feasible sequence of tasks which meets the precedence constraints is considered for illustration. Let, the particle be, 1***-3-2-4-5-6-7-9-8-10-11.*** 11 task and 4 workstation problem is considered for the illustration. The robot energy consumption and performance time details are presented in Table 3. The sequence meets the precedence constraints shown in Figure 2.

Step 1. E_0 is calculated and it is found to be 35. Initial value of E_0 is the mean of minimum energy consumption of the robots for the tasks. The initial energy consumption (E_0) of the assembly line is calculated using Equation 3

$$E_0 = [\sum_{i=1}^{N_a} \min_{1 \le h \le N_r} e_{hi} / N_w] \quad (3)$$

Here N_a is the total number of tasks, N_w is the total number of workstations, N_r is the total number of robots and e_{hi} is the energy consumption by robot h to perform task i.

Step 2. First station is opened and procedure tries to allocate the tasks from either side of the sequence, if one or more robot could perform the allocated tasks within E_0. Procedure tries for maximal allocation of tasks within this E_0. Tasks are allocated to workstations by moving forward and backward through the precedence diagram.

Step 3. Open the next workstation and try allocating the remaining tasks. Repeat steps 2 and 3 until all task are assigned.

Step 4. If the procedure could not assign all the tasks within the initial E_0, increment E_0 by 'one' and repeat the Step 2 and 3 until all tasks get assigned to the workstation.

Step 5. Each workstation is allotted with certain set of tasks. The best available robot which performs the allotted tasks with minimum energy consumption is selected and assigned.

Step 6. The workstation time is calculated for the allocation of tasks and robots.

Step 7. The workstation with the maximum workstation time is the cycle time of the allocation.

Step 8. Line Efficiency is calculated using Equation 4 using the workstation times.

The line efficiency is calculated using Equation 4.

$$LE = \frac{\sum_{k=1}^{N_w} S_k}{N_w * \mathrm{C}} * 100 \qquad (4)$$

Here $\underline{S_k}$ is the workstation time, N_w is total number of workstations and C is the cycle time.

In the given example, procedure tries to find the optimal assignment within this E_0, it is not possible to find a solution for four stations as shown in Figure 3. From Figure 3 it could be understood that tasks 5 and 6 are left to be unassigned within this E_0. E_0 is incremented until 48 to achieve complete allocation of tasks and robots. Complete allocation is shown in Figure 4. Shaded portion in Figure 4 shows the robot selected for performing the allotted tasks for each workstation. From the allocation, the workstation times are calculated using robot performance time.

Workstation 1 Time: 121 (Robot 2, Tasks-1, 11, 10, Time-37+46+38)

Workstation 2 Time: **162** (Robot 3, Tasks-3, 8, 2, Time-38+34+90)

Workstation 3 Time: 98 (Robot 4, Tasks-4, 9, 5, Time-40+33+25)

Workstation 4 Time: 128 (Robot 1, Tasks-6, 7, Time-77+51)

Line Efficiency: (121+162+98+128) / (4*162) = 0.7854= 78.54%.

Here number of workstations is 4 and the cycle time is 162. The line efficiency is calculated and found to be 78.54%.

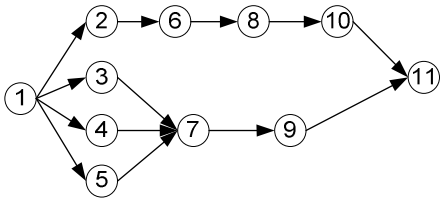

Figure 2 Precedence Graph of 11 task problem

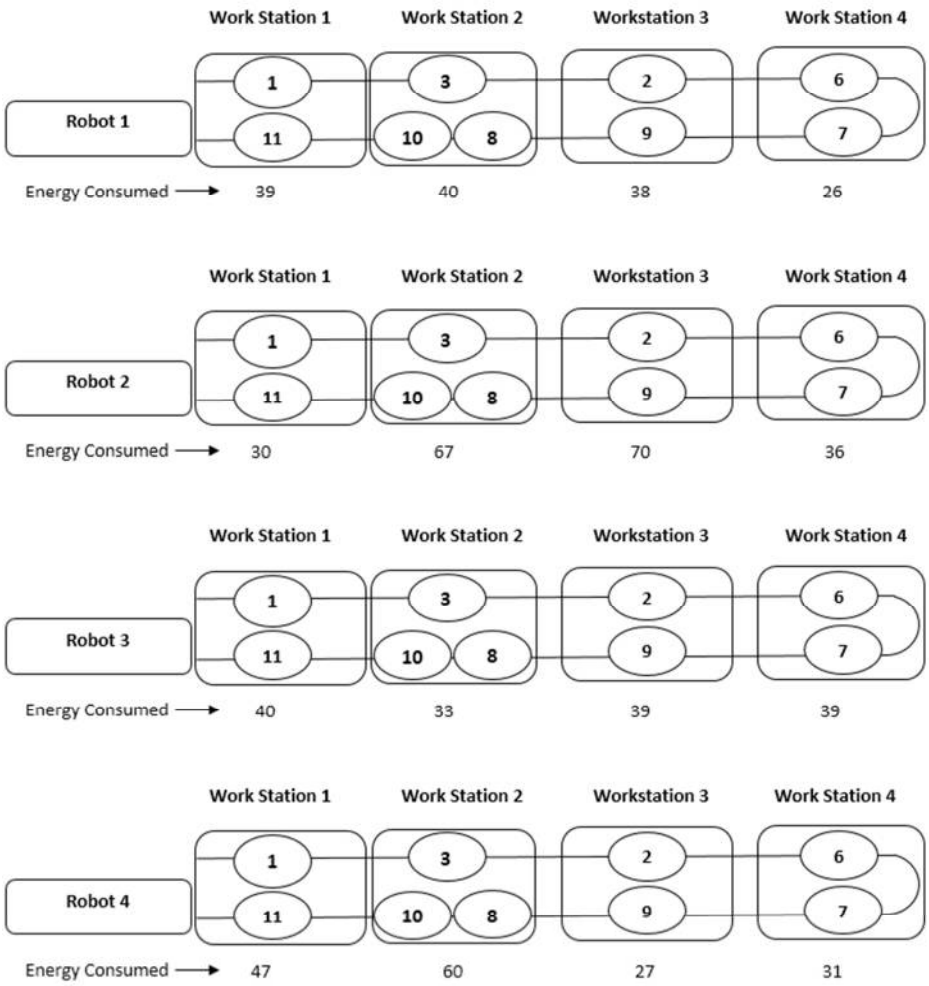

Figure 3 Task and robot assignment for initial cycle time (Unit of energy consumption is kilojoules)

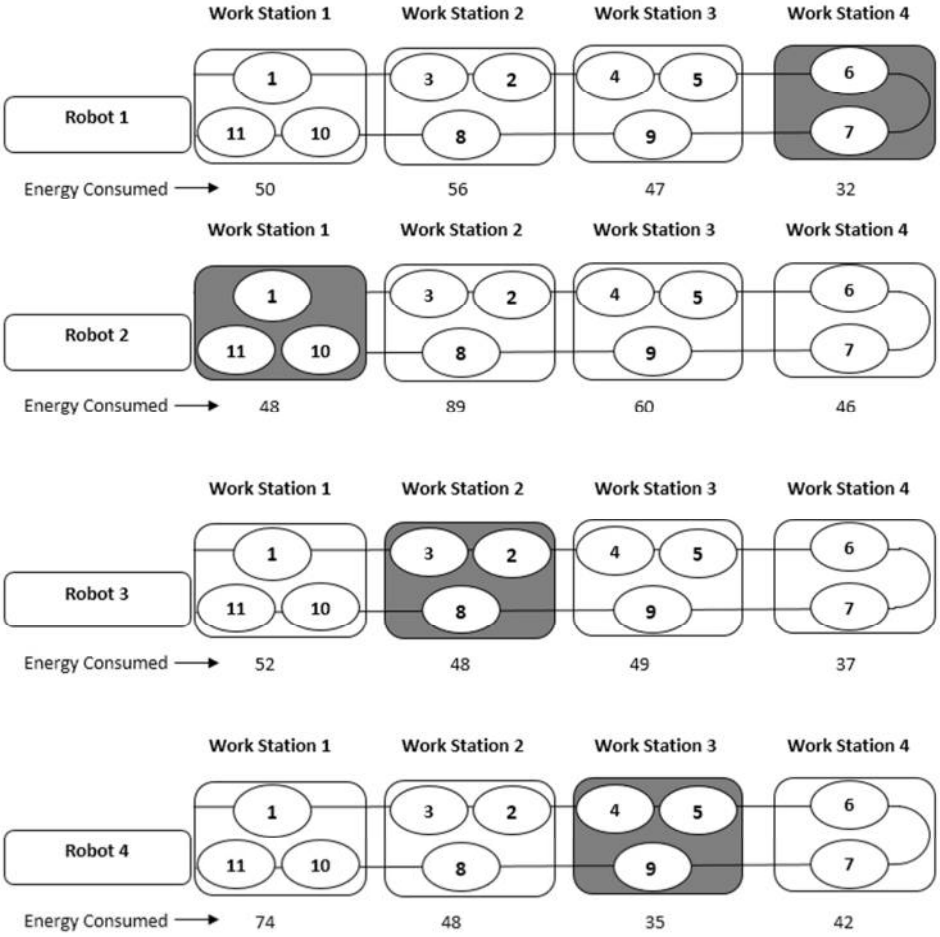

Figure 4 Final Assignment of tasks and allocation of robots ((Unit of energy consumption is kilojoules)

Table 3. Energy consumption and performance time for 11 tasks by 4 robots

Tasks	Energy Consumption (kilojoules)				Performance Time Units			
	Robot 1	Robot 2	Robot 3	Robot 4	Robot 1	Robot 2	Robot 3	Robot 4
1	20	15	15	17	81	37	51	49
2	27	40	27	15	109	101	90	42
3	16	32	11	18	65	80	38	52
4	13	16	27	14	51	41	91	40
5	23	14	10	9	92	36	33	25
6	19	26	25	25	77	65	83	71
7	13	20	12	17	51	51	40	49
8	13	17	10	15	50	42	34	44
9	11	30	12	12	43	76	41	33
10	11	18	12	27	45	46	41	77
11	19	15	25	30	76	38	83	87

3.2 Differential Evolution

Differential evolution (DE) is an evolutionary algorithm proposed by (Storn and Price, 1997) for solving optimization and engineering problems. Different types of numerical optimization problems have been tested using DE and it could be found that DE outperforms other evolutionary algorithms. DE has only three parameters controlling the search process explained later in the paper. DE is widely utilized to solve many real-world problems like job shop scheduling and engineering design optimization due to its simplicity (Wang et al., 2014).

DE is very similar to genetic algorithm the main differences are on mechanism of mutation and crossover (Nearchou, 2005). DE starts by generating a random set of population composed of 'target vectors'. This population undergoes evolution in a form of natural selection. In each generation, mutation, crossover and selection operators are applied for producing new and better population with higher quality individuals. In every iteration, each target vector undergoes a mutation operation and a set of donor vector is created. Crossover operation between target and donor vector is done to create a set of trial vectors. Compare the fitness value of every trial vector with corresponding target vector. The vector which gives the better objective function value is copied to the next generation. The evolutionary process terminates after a number of generations is met. Pseudo code of DE is given Figure 5

```
procedure DE
input (problem data, DE parameters, termination condition)
Generate randomly a population Φ of N_p (Target vectors);
 Evaluate (Φ); //Refer Section 3.1.3
while termination condition not satisfied do
    for i = 1 to N_p do
        Select the next target vector X_j from Φ;
        Choose randomly 3 vectors X_a, X_b, X_c from Φ;
        Generate a mutant vector u using the Equation 5;
        Crossover the mutant and target vector to generate a trial vector y ;
        Evaluate the trial vectors f(y);
        if f(y) > f(X_j) then
            Replace y with X_j ;
    endfor;
    Save best-so far vector to X*
 endwhile
 Return X*
```

Figure 5 Pseudo code of DE

3.2.1 Initial Population

DE procedure starts with the initial set of population called as 'target vectors'. Each member (vector) of this population encodes a potential solution for the problem. The vector represents a sequence of numbers (tasks) arranged such a way that it meets the precedence relationship. The same procedure used for creating initial set of population for PSO is used in DE also.

3.2.2 Mutation

Mutation operator is the prime operator of DE and it is the implementation of this operation that makes DE different from other Evolutionary algorithms. The mutation process at each generation begins by randomly selecting three individuals in the population. A population of donor vectors is created by perturbing the population of target vectors. Perturbation is performed by adding the difference between two randomly selected target vectors to a third target vector as shown in Equation 5.

$$y_{ig} = x_{r1,g} + F(x_{r2,G} - x_{r3,G}), where\ i\ = 1,...5 \qquad (5)$$

F is known as the mutation scaling factor. An example is shown to explain how mutation process works in a RALB problem. Let the three vectors be:

$x_{r1,G}$={1,2,6,3,4,5,7,8,10,9,11} $x_{r2,G}$={1,2,3,4,5,6,7,8,9,10,11}and

$x_{r3,G}$={1,2,3,6,5,4,7,8,10,9,11}, F=0.5

y_{ig}={1,2,6,3,4,5,7,8,10,19,11}+0.5x{1,2,3,4,5,6,7,8,9,10,11}-{1,2,3,6,5,4,7,8,10,9,11}

The pairs of transpositions to get $x_{r3,G}$ from $x_{r2,G}$ are identified. Mutation factor is applied to select the number of pairs and these selected pairs is used to transposition the values in $x_{r1,g}$

y_{ig}={1,2,6,3,4,5,7,8,10,19,11}+0.5x(3,5)(8,9)={1,2,6,3,4,5,7,8,10,9,11}+(8,9)={1,2,6,3,4,5,7,8,9, 10,11}

3.2.2 Crossover

Once the mutation phase is complete, the crossover process is activated. Set of trial vectors are created by choosing between the donor vector and target vector. Crossover is done for a set of selected vectors in the population. Number of vectors for crossover is selected based on crossover rate C_R. Trail vectors are generated by using OX operator (Order Crossover) proposed by Davis (1985).

OX Operators works as follows:

- Select a subsection of task sequence from target vector at random.
- Produce a proto-trail vector by copying the substring of task sequence into the corresponding positions.
- Delete the tasks that are already in the substring from the donor vector. The resulted sequence of tasks contains tasks that the proto-trial vector needs.
- Place the tasks into the unfixed positions of the proto-trial from left to right according to the order of the sequence in the donor vector.

An example explaining this method is given in Figure 6.

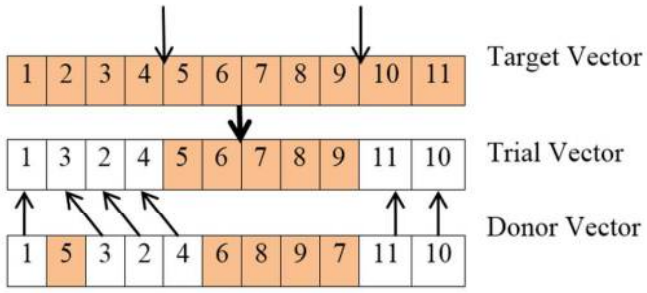

Figure 6 Illustration of the OX operator

A reordering procedure used by (Levitin et al., 2006) is also incorporated to make the vectors feasible if the created vector does not meet the precedence constraints

3.2.2 Selection

The selection scheme of DE also differs from that of other evolutionary algorithms (Ali et al., 2009). The population for the next generation is selected from the individual in current population and its corresponding trial vector. Target vector competes with their corresponding trial vector to be selected on to the next generation/iteration. The vector with the better fitness value is copied to the next generation. In this research, selection is based on the objective of better efficiency of the assembly line. The rule of selection is according to the following rule:

$$x_{i,\mathrm{G}+1} = \begin{cases} \mathrm{Z_{i,G}} & if\ f(Z_{i,G}) > f(x_{i,G}) \\ 0, & otherwise \end{cases} \qquad (6)$$

The algorithm is terminated if the iteration approaches a predefined criteria, usually a sufficiently good fitness or in this case, a predefined maximum number of iterations (generations) is used.

3.2.3 Fitness Value

The fitness value to be evaluated in this research is to maximize the line efficiency by minimizing the energy consumption of the assembly line. Refer Section 3.1.3 for the procedure of fitness evaluation adopted in this paper.

4. Results and Discussion

To demonstrate the effectiveness of the two proposed algorithms, computational experiments are conducted. The following section describes the experiments conducted.

4.1 Dataset for computational experiments

Scholl (1995) generated thirty two problems with tasks ranging from 25 to 297 for simple assembly line balancing problems using eight representative precedence graphs, which are available in http://www.assembly-line-balancing.de/. Gao et al. (2009) generated 32 test problems for RALB using this eight precedence graphs by including the robot performance times. Using the robot performance time and power consumption of the robots, energy consumption of the robots are calucalted. Power consumption of the robots are randomly generated. Power consumption deatilas of small datasets are shown in Appendix A and for large datasets it is shown in Appendix B. These datasets are used in this research to evaluate the two proposed models. The energy consumption of a task *i* by robot *h* is calculated as follows:

$$E = P_h \times t_{ih} \tag{7}$$

4.2 Parameter Selection for PSO and DE

Performance of PSO and DE mainly relies on the parameters selected. Parameters are selected based on the tests conducted in order to get a satisfactory solution quality in an acceptable time span. Influence of each parameter on the solution quality is tested. Three datasets of different task size are chosen to find the best combination of parameters.

Following are the parameters tested and used in PSO:

Stopping Conditions: The proposed PSO and DE algorithm is terminated if the maximum number of generation reaches a predefined criteria, usually a sufficiently good fitness or in this case, a predefined maximum number of iterations (generations) is used. Different stopping

conditions are tested such as 5, 10, 15, 25 and 30 and best solution could be obtained when number of generation is 25 for both PSO and DE. After 25th iteration the proposed algorithm started obtaining same solution. Figure 7 shows the performance of PSO based on the stopping condition and Figure 8 shows the performance of DE based on the stopping condition. Three problems are taken into consideration for generating the graph. Problems are 35-5, 89-8, 111-13 where 35, 89, 111 are the number of tasks whereas 5, 8, 13 indicates the number of robots and work stations available to perform the tasks.

Acceleration coefficients: Different combinations of acceleration coefficients are tested. Table 4 shows the different combinations of c_1, c_2 and c_3 tested. The same problem set considered for stopping condition is considered here and the performance of the PSO algorithm based on the combination is evaluated. Figure 9 shows the performance of the algorithm based on different combinations of acceleration coefficients for three sample problems from the literature. From the experiments it could be found that Group D with the c_1=1, c_2=2, c_3=2 could obtain the best solution. This combination of acceleration coefficient is used to evaluate all thirty two test problems.

Table 4. Selection Group for Acceleration Coefficients

Group	c1	c2	c3
A	1	1	1
B	1	1	2
C	1	2	1
D	1	2	2
E	2	1	1
F	2	1	2
G	2	2	1
H	2	2	2

Swarm Size: The swarm size used in PSO and DE algorithms is 25. By conducting trial and error experiments on sample datasets it is found that swarm size of 25 could yield a better performance value in a reasonable computational time. The other reason for fixing the swarm size as 25 is that generating and maintain feasible strings for assembly line scheduling problems is also to be taken care of. During evolution the maintenance of feasibility by way of adopting some repair scheme is also required. Because of these factors the swarm size is fixed as 25 in this research.

Crossover rate: Crossover rate (C_R) reflects the probability with which the trial individual inherits the actual individual's genes (Feoktistov, 2006). If the C_R value is relatively high, this

will increase the population diversity and improve the convergence speed (Mohamed et al., 2012). Different levels of crossover rate (0.3, 0.5, 0.7, and 0.9) are tested. Three problems of different task size are evaluated using the different levels of crossover. Best solution could be obtained when the C_R value is 0.9. Figure 10 shows the performance of the algorithm based on the crossover rate.

Mutation Factor: F is a mutation scaling factor of the difference vector (Equation 5). These parameters help to control the evolving rate of the population. F is chosen to be a value in [0, 2] for original DE algorithm (Storn and Price, 1997). When small F values are used it could lead to premature convergence and high values can slow down the search (Mohamed et al., 2012). From the literature review it could be found that mutation factor 0.5 is better and this value is used for solving all the thirty two problems.

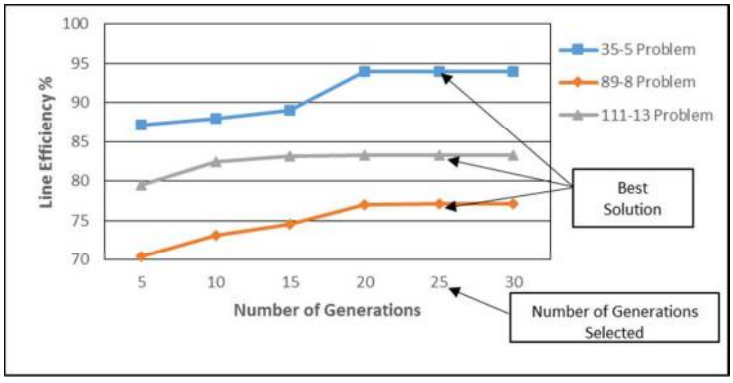

Figure 7 Comparison of PSOs' performance in terms of stopping condition

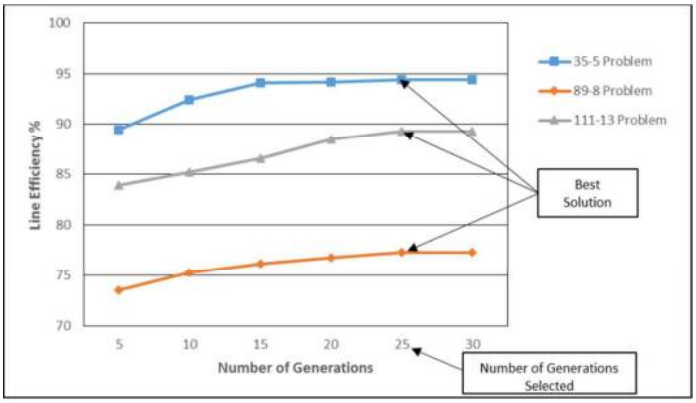

Figure 8 Comparison of DEs' performance in terms of stopping condition

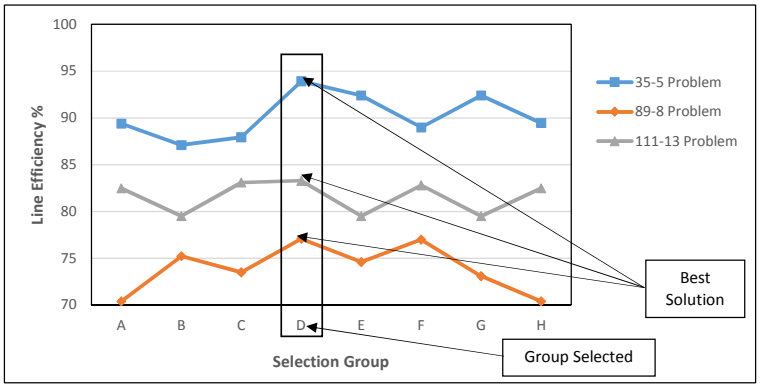

Figure 9 Selection of acceleration coefficients based on the performance of PSO algorithm

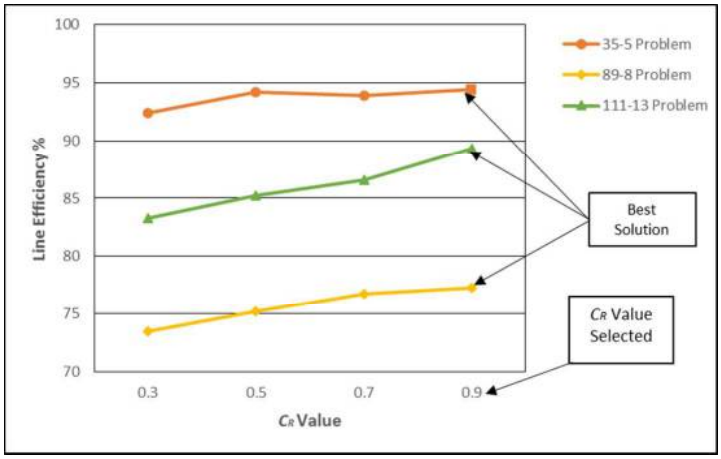

Figure 10 Selection of Crossover rate (C_R) value based on the performance of DE algorithm

4.3 Performance Comparison of PSO and DE

All the 32 test problems are evaluated using the proposed PSO and DE algorithm. The non-deterministic nature of the algorithm and problem makes it necessary to run same problem multiple times. Each problem is run ten times and most of the runs converged to the same solution for each of the problems. All problems are run ten times and most of the runs converged to the same solution for most of the problems. The results obtained by evaluating 32 test problems are presented in Table 5. Column I and II shows the number of tasks and robots (workstations) considered for the evaluation. The energy consumption, cycle time, line efficiency (objective function) and the computational time of the proposed two models are reported.

The problems are classified into two categories based on the tasks size: small (up to 89 tasks) and large (above 100 tasks). Proposed algorithms allocate robots to the workstation by allocating

the robots and tasks which minimizes the total energy consumption in each workstation. Using the solution obtained, the workstation time and cycle time of the assembly line are calculated. The workstation time and cycle time is used for calculating the line efficiency of the assembly line. Thus line efficiency is obtained by minimizing the total energy consumption. It could be seen from the experiments conducted, 28 out of 32 datasets could obtain better line efficiency for DE when compared to PSO. DE could obtain improvement of 5.4% in line efficiency when compared with PSO for small datasets. In case of large datasets, the obtained improvement was 1.7%. Percentage could be low for large datasets because both PSO and DE could reach solutions which are nearer to each other.

Table 5. Results of the 32 RALB problems

Tasks	Work Station/ Robots	Proposed Algorithm (PSO)				Proposed Algorithm (DE)			
		Total Energy Consumption (kilojoules)	Cycle Time	Line Efficiency (%)	CPU Time ('s')	Total Energy Consumption (kilojoules)	Cycle Time	Line Efficiency (%)	CPU Time ('s')
25	3	488	561	85.6	7.2	493	496	98.6	5.8
	4	336	334	87.7	8.7	341	320	90.7	6.9
	6	358	230	85.6	12.3	363	230	87.3	9.2
	9	242	148	72.7	16.5	244	154	72.6	13.2
35	4	1010	535	79.2	19.8	1045	477	82.4	16.3
	5	855	397	94.1	22.5	909	380	94.4	19.4
	7	983	307	87.2	29.4	998	302	87.5	24.4
	12	653	125	81.2	33.5	681	122	86.4	29.3
53	5	2638	610	77.08	38.5	2680	464	96.3	31.2
	7	1959	330	89.7	41.8	1970	326	95.8	37.4
	10	2133	260	83.6	44.6	2150	249	91.3	41.3
	14	1990	183	80.2	49.2	2003	183	83.6	43.9
70	7	4256	640	94.6	68.1	4364	656	95.7	62.6
	10	3013	298	87.7	69.8	3050	294	89.3	66.3
	14	3700	272	72.4	75.4	3739	266	78.9	71.2
	19	3063	170	79.0	87.2	3223	176	81.0	82.4
89	8	4926	558	77.1	89.5	4991	545	77.3	85.9
	12	5496	426	80.2	96.1	5537	424	82.7	89.2
	16	4753	285	81.4	99.2	4798	269	90.7	91.3
	21	4135	200	82.7	111.3	4158	201	84.9	123.4
111	9	7138	696	95.1	244.9	7172	688	90.4	263.2
	13	6818	378	83.3	259.7	7078	377	89.3	281.9
	17	6688	266	89.9	320.5	6804	265	92.5	345.1
	22	6518	217	83.9	355.4	6538	223	80.8	379.2
148	10	9798	671	94.8	449.8	9798	664	96.4	469.7
	14	10395	458	90.2	530.2	10732	521	85.7	545.1
	21	10235	394	83.8	613.1	9894	318	84.8	643.5
	29	8186	200	86.5	645.3	8296	198	88.3	679.9
297	19	24313	696	88.3	1592.2	24320	695	88.8	1642.1
	29	24302	592	67.0	1713.8	24456	500	78.9	1803.3
	38	22037	350	85.3	1782.5	22585	357	85.9	1892.6
	50	20647	255	86.1	1822.3	20888	256	86.4	1942.4

But, the line efficiency of the solution obtained by DE is better than PSO. From the table, cycle time obtained for both DE and PSO are reported along with the energy consumption of the assembly line. It could be seen that cycle time is lower in case of DE solution for 25 out of 32 problems. When comparing the energy consumption amongst the two models, it could be seen that PSO is getting lower energy consumption compared to that of DE for 31 out of 32 problems addressed here.

The algorithm structured based on PSO provides reasonable quality assignment of tasks and robots to workstations for small size problems in practical computational time but DE algorithm could obtain the solution for the same set of solution at a faster rate. In case of large datasets, DE algorithm takes more computational time than PSO. The computational time for DE could be on the higher side when compared to PSO due to repeated fitness value evaluation in case of selection operation. Though the results of DE show that DE is better for larger size problems in terms of line efficiency, its robustness and computational efficiency can be improved by fine tuning the parameters.

4.4 Statistical Analysis: Pairwise comparisons using Wilcoxon Signed-rank Test

The objective of the statistical analysis is to compare the performances of the proposed algorithms. Pairwise comparisons are the simplest statistical tests those are useful to compare the performance of two algorithms when applied to a common set of problems. Wilcoxon Signed-rank test is one among the tests used for pairwise comparisons. This test is more sensitive than the *t-test*. It assumes commensurability of differences, but only qualitatively: greater differences still count for more, which is probably desired, but the absolute magnitudes are ignored. From the statistical point of view, the test is safer since it does not assume normal distributions (Derrac et al., 2011). Hence, Wilcoxon Signed-rank test is conducted to test the performance of PSO and DE for the three objective functions considered. This test provides the p-values for the pairwise comparisons. The details of p-values obtained and other parameters used to conduct the test are shown in Table 6. From the statistical results it can be concluded that DE performs better than PSO for cycle time and line efficiency. When comparing the performance in terms of energy consumption PSO performs better than DE.

Table 6. Wilcoxon Signed-rank test results

	PAIRWISE COMPARISONS of PSO and DE Parameters used: α = 0.05, two-tailed test		
	PSO-DE (Energy consumption)	PSO-DE (Cycle Time)	**PSO-DE (Line Efficiency)**
P-values	0	0.01278	0.00016
n	31	30	32
Critical Value	147	137	159
Result	PSO is better than DE	DE is better than PSO	DE is better than PSO

5. Conclusion

In this paper a Robotic Assembly Line Balancing (RALB) problem is addressed with an objective of maximizing the line efficiency by minimizing the energy consumption. Reduction of energy consumption is given importance in manufacturing sectors due to sequence of serious environmental impacts and rising energy cost. In the present day contest it is very important in creating an eco-friendly manufacturing system by minimizing the energy consumption. This paper is an important addition to the literature where the majority of robotic assembly line balancing researches till date focused on objectives of minimizing cycle time and minimizing cost while energy-related optimization is ignored. Two evolutionary algorithms is proposed to solve the problem with an objective of maximizing the line efficiency by minimizing the energy consumption. Thirty two datasets available in the literature has only time and precedence information. The energy data is embedded into the existing datasets for the experiments conducted. The computational experiments are conducted on the two proposed algorithms in this paper. From the results obtained through these algorithms it could be observed that DE based algorithm could obtain better line efficiency than PSO based algorithm. Energy consumption and cycle time of the problems are also reported. It could be observed that DE based model could obtain solution in lesser computational time than PSO for large datasets. But in case of large datasets PSO performs faster than DE. This could be due to repeated fitness evaluation during selection process. In future work, the algorithm could be tested on by considering a specific time horizon, where the factors such as maintenance operation and effect of failures of the resources in the system could be included. Other evolutionary algorithms, as an alternate to PSO and DE, may be attempted for solving the same problem.

References

Ali, M., Pant, M. & Abraham, A. 2009. Simplex differential evolution. *Acta Polytechnica Hungarica,* Vol.6**,** No.5, pp.95-115.

Baykasoğlu, A. & Özbakır, L. 2007. Stochastic U-line balancing using genetic algorithms. *The International Journal of Advanced Manufacturing Technology,* Vol.32**,** No.1-2, pp.139-147.

Bukchin, J., & Tzur, M. 2000. Design of flexible assembly line to minimize equipment cost. *IIE transactions*, Vol.32, No.7,pp.585-598.

Dai, M., Tang, D., Giret, A., Salido, M. A. & Li, W.D. 2013. Energy-efficient scheduling for a flexible flow shop using an improved genetic-simulated annealing algorithm. *Robotics and Computer-Integrated Manufacturing,* Vol.29**,** No.5, pp.418-429.

Daoud, S., Chehade, H., Yalaoui, F. & Amodeo, L. 2014. Solving a robotic assembly line balancing problem using efficient hybrid methods. *Journal of Heuristics*, Vol.20,No.3,pp.235-259.

Davis, L. Applying adaptive algorithms to epistatic domains. 1985. *Proceedings of the 9th international Joint Conference on Artificial Intelligence*,Vol.1, pp.162-164.

Derrac, J., García, S., Molina, D., & Herrera, F. 2011. A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation*, Vol.1, No.1,pp.3-18.

Erel, E., Sabuncuoglu, I. & Aksu, B. 2001. Balancing of U-type assembly systems using simulated annealing. *International Journal of Production Research,* Vol.39, No.13, pp**.**3003-3015.

Feoktistov, V. 2006. Differential evolution, *Springer*.

Fysikopoulos, A., Anagnostakis, D., Salonitis, K. & Chryssolouris, G. 2012. An empirical study of the energy consumption in automotive assembly. *Procedia CIRP,* Vol.3**,** pp.477-482.

Gao, J., Sun, L., Wang, L. & Gen, M. 2009. An efficient approach for type II robotic assembly line balancing problems. *Computers & Industrial Engineering,* Vol.56**,** No.3, pp.1065-1080.

Gutjahr, A. L. & Nemhauser, G. L. 1964. An algorithm for the line balancing problem. *Management Science,* Vol.11, No.2, pp. 308-315.

Kaveh, A. & Talatahari, S. 2011. Hybrid charged system search and particle swarm optimization for engineering design problems. *Engineering Computations,* Vol. 28, No.4, pp.423-440.

Kennedy, J. & Eberhart, R. Particle swarm optimization.1995. *Proceedings of IEEE International Conference on Neural Networks*, Vol.4, pp. 1942 – 1948.

Kumar, D. M. 2013. Assembly Line Balancing: A Review of Developments and Trends in Approach to Industrial Application. *Global Journal of Researches In Engineering,* Vol.13, No.2.

Levitin, G., Rubinovitz, J. & Shnits, B. 2006. A genetic algorithm for robotic assembly line balancing. *European Journal of Operational Research,* Vol.168,No.3,pp.811-825.

Luo, H., Du, B., Huang, G. Q., Chen, H. & Li, X. 2013. Hybrid flow shop scheduling considering machine electricity consumption cost. *International Journal of Production Economics,* Vol.146,No.2,pp.423-439.

Martí, R. & Reinelt, G. 2011. The linear ordering problem: exact and heuristic methods in combinatorial optimization, Vol.175, *Springer*.

Miltenburg, G. & Wijngaard, J. 1994. The U-line line balancing problem. *Management Science,* Vol.40**,** No.10,pp.1378-1388.

Mohamed, A. W., Sabry, H. Z. & Khorshid, M. 2012. An alternative differential evolution algorithm for global optimization. *Journal of Advanced Research,* Vol.3,No,2,pp.149-165.

Mouzon, G., Yildirim, M. B. & Twomey, J. 2007. Operational methods for minimization of energy consumption of manufacturing equipment. *International Journal of Production Research,* Vol.45**,** No.18-19,pp.4247-4271.

Mukund Nilakantan, J. & Ponnambalam, S.G. 2012 An efficient PSO for type II robotic assembly line balancing problem. *Proceedings of IEEE International Conference on Automation Science and Engineering (CASE)*, pp.600-605.

Nicosia, G., Pacciarelli, D. & Pacifici, A. 2002. Optimally balancing assembly lines with different workstations. *Discrete Applied Mathematics*, Vol.118,No.1-2,pp.99-113.

Nearchou, A. C. 2005. A differential evolution algorithm for simple assembly line balancing. *Proceedings of 16th Int. Federation of Automatic Control (IFAC) World Congress*, Prague.

Ponnambalam, S.G, Aravindan, P. & Naidu, G. M. 2000. A multi-objective genetic algorithm for solving assembly line balancing problem. *The International Journal of Advanced Manufacturing Technology,* Vol.16,No.5,pp.341-352.

Rashid, M. F. F., Hutabarat, W. & Tiwari, A. 2012. A review on assembly sequence planning and assembly line balancing optimisation using soft computing approaches. *The International Journal of Advanced Manufacturing Technology,* Vol.59, No.1-4, pp.335-349.

Rameshkumar, K., Suresh, R. K., &Mohanasundaram, K. M. 2005. Discrete particle swarm optimization (DPSO) algorithm for permutation flowshop scheduling to minimize makespan. *Proceedings of Advances in Natural Computation, Springer Berlin Heidelberg*,pp.572-581

Ratnaweera, A., Halgamuge, S. & Watson, H. C. 2004. Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients. *Evolutionary Computation, IEEE Transactions,* Vol.8,No.3,pp.240-255.

Rubinovitz, J. & Bukchin, J. 1991. *Design and balancing of robotic assembly lines*, Society of Manufacturing Engineers.

Rubinovitz, J., Bukchin, J. & Lenz, E. 1993. RALB–A heuristic algorithm for design and balancing of robotic assembly lines. *CIRP Annals-Manufacturing Technology,* Vol. 42, No.1, pp.497-500.

Salveson, M. E. 1955. The assembly line balancing problem. *Journal of Industrial Engineering,* Vol.6, pp.18-25.

Scholl, A. 1995. Data of assembly line balancing problems. Darmstadt Technical University, Department of Business Administration, Economics and Law, Institute for Business Studies *(BWL)*.

Scholl, A. & Becker, C. 2006. State-of-the-art exact and heuristic solution procedures for simple assembly line balancing. *European Journal of Operational Research,* Vol. 168, No.3, pp.666-693.

Shrouf, F., Ordieres-Meré, J., García-Sánchez, A. & Ortega-Mier, M. 2014. Optimizing the production scheduling of a single machine to minimize total energy consumption costs. *Journal of Cleaner Production,* Vol. 67, pp.197-207.

Storn, R. & Price, K. 1997. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization,* Vol. 11, No.4, pp.341-359.

Wang, G.-G., Gandomi, A. H., Yang, X.-S., Alavi, A. H. & Owen, D. 2014. A Novel Improved Accelerated Particle Swarm Optimization Algorithm for Global Numerical Optimization. *Engineering Computations,* Vol. 31, No.7, pp.1198-1220.

Yoosefelahi, A., Aminnayeri, M., Mosadegh, H. & Ardakani, H. D. 2012. Type II robotic assembly line balancing problem: An evolution strategies algorithm for a multi-objective model. *Journal of Manufacturing Systems,* Vol.31, No.2, pp.139-151.

Appendix A

Robot Power Details for Small Size data (Power in kW)

Problem	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15	R16	R17	R18	R19	R20	R21
11-4	0.25	0.4	0.3	0.35	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
25-3	0.4	0.35	0.3	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
25-4	0.25	0.4	0.3	0.3	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
25-6	0.3	0.4	0.4	0.3	0.3	0.35	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
25-9	0.2	0.3	0.25	0.4	0.35	0.4	0.25	0.3	0.3	--	--	--	--	--	--	--	--	--	--	--	--
35-4	0.5	0.8	0.8	0.9	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
35-5	0.7	0.5	0.8	0.9	0.5	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
35-7	0.9	0.8	0.5	0.9	0.7	0.8	0.5	--	--	--	--	--	--	--	--	--	--	--	--	--	--
35-12	0.9	0.8	0.6	0.8	0.9	0.5	0.5	0.6	0.7	0.5	0.7	0.9	--	--	--	--	--	--	--	--	--
53-5	1.1	1.2	1.5	0.9	1.3	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
53-7	1.5	0.9	1.2	1.1	1.2	1.3	1.5	--	--	--	--	--	--	--	--	--	--	--	--	--	--
53-10	1.1	0.9	1.2	1.5	1.3	0.9	1.2	1.4	1.1	0.9	--	--	--	--	--	--	--	--	--	--	--
53-14	0.9	1.2	1.5	1.1	1.4	0.9	1.2	0.9	1.5	1.3	1.2	1.4	1.3	0.9	--	--	--	--	--	--	--
70-7	1.4	1.4	1.7	1.5	1.3	1.5	1.4	--	--	--	--	--	--	--	--	--	--	--	--	--	--
70-10	1.2	1.6	1.1	1.7	1.2	1.5	1.4	1.8	1.6	1.7	--	--	--	--	--	--	--	--	--	--	--
70-14	1.8	1.4	1.5	1.7	1.1	1.4	1.5	1.3	1.8	1.6	1.1	1.7	1.4	1.2	--	--	--	--	--	--	--
70-19	1.5	1.8	1.1	1.7	1.3	1.6	1.6	1.2	1.8	1.3	1.7	1.6	1.3	1.1	1.3	1.1	1.1	1.5	1.7	--	--
89-8	1.1	1.8	1.5	1.2	1.9	1.4	1.2	1.6	--	--	--	--	--	--	--	--	--	--	--	--	--
89-12	1.8	1.2	1.4	1.2	1.4	1.4	1.6	1.7	1.9	1.5	1.1	1.5	--	--	--	--	--	--	--	--	--
89-16	1.5	1.6	1.7	1.4	1.6	1.7	1.4	1.5	1.8	1.4	1.2	1.5	1.6	1.4	1.3	1.3	--	--	--	--	--
89-21	1.5	1.1	1.7	1.8	1.6	1.4	1.7	1.8	1.3	1.9	1.9	1.5	1.4	1.3	1.6	1.6	1.4	1.9	1.6	1.1	1.4

Appendix B

Robot Power Details for Large Size data (Power in kW)

Problem	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15	R16	R17	R18	R19	R20	R21	R22
111-9	2.1	1.7	2.4	1.8	1.2	2.2	1.5	1.8	2.3	--	--	--	--	--	--	--	--	--	--	--	--	--
111-13	1.8	2.1	1.5	2.2	2.3	2.4	1.9	1.7	1.9	2.2	2.1	2.3	2.4	--	--	--	--	--	--	--	--	--
111-17	1.7	1.9	1.6	2.2	2.1	1.8	1.6	1.5	2.1	1.6	2.3	1.9	1.8	2.4	1.7	2.2	2.1	--	--	--	--	--
111-22	2.1	1.5	1.5	1.6	1.7	1.6	2.1	2.4	1.8	1.9	1.5	1.7	1.6	2.2	1.8	1.6	2.3	2.1	1.6	1.7	2.2	2.4
148-10	2.1	2.2	1.8	2.4	1.6	1.5	2	2.3	2.4	2.5												
148-14	2.2	2.0	2.1	2.4	1.8	2.4	1.7	1.5	1.7	2.1	2.2	2.0	1.7	2.3								
148-21	2.4	1.5	1.6	2.4	2.4	1.5	1.9	1.6	2.0	2.0	2.1	2.2	2.0	2.1	1.5	1.9	2	2.1	1.8	2.1	2	
297-19	2.8	2.2	2.3	2.4	2.6	2.4	1.9	2	2.2	2.3	1.9	2.6	3	2.7	2.5	2.5	2.2	2.1	2.7			

Due to space constraints power details of four problems are not given in the table

148-29: (2.2, 2.3, 1.6, 1.6, 2.4, 2.3, 2.0, 1.6, 2.2, 1.6, 1.6, 2.1, 1.7, 1.6, 1.7, 1.7, 2.4, 2.2, 2.2, 1.8, 1.8, 1.5, 2.1, 1.9, 1.8, 1.9, 2.4, 1.7, 1.5)

297-29: (2.3, 1.9, 2.3, 2.1, 2.1, 2.0, 2.9, 2.1, 2.2, 2.8, 2.7, 2.5, 2.1, 2.8, 2 .1, 3.0, 2.4, 2.7, 2.9, 2.2, 2.2, 2.1, 2.1, 2.8, 2.1, 2.7, 2.6, 2.7, 2.9)

297-38: (2.3, 2.3, 1.8, 2.0, 2.9, 1.8, 2.4, 2.3, 2.9, 2.0, 2.8, 2.0, 2.7, 2.9, 2.7, 2.2, 2.3, 2.5, 2.6, 2.5, 2.1, 1.8, 2.1, 2.9, 2.5, 2.8, 1.9, 1.9, 2.1, 2.7, 2.4, 2.2, 1.9, 2.6, ,2.7, 3.0, 2.5, 2.8)

297-50: (2.4, 2.3, 2.5, 2.4, 2.2, 2.0, 2.5, 2.3, 2.3, 2.4, 2.0, 2.7, 2.3, 2.6, 2.4, 2.4, 2.5, 2.1, 2.3, 1.9, 2.0, 2.3, 2.9, 2.2, 2.4, 2.2, 2.7, 1.8, 2.9, 1.8, 1.8, 1.8, 2.4, 2.3, ,2.2, 2.2, 1.9, 1.8, 2.6, 2.1, 2.6, 2.5, 2.6, 2.5, 1.8, 2.8, 2.3, 2.4, 2.9, 2.7)