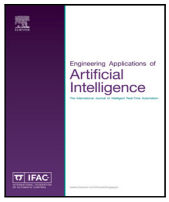
Engineering Applications of Artificial Intelligence 114 (2022) 105112

Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

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



# Solving the energy-efficient Robotic Mixed-Model Assembly Line balancing problem using a Memory-Based Cuckoo Search Algorithm

Lakhdar Belkharroubi *, Khadidja Yahyaoui

*University Mustapha Stambouli, Mascara, 29000, Algeria*

ARTICLE INFO

*Keywords:*
Bio-inspired algorithm
Robotic assembly lines
Energy consumption
Artificial intelligence
Memory-based algorithm

ABSTRACT

Minimizing energy consumption is an important issue in robotic assembly lines where a set of robots are assigned to a set of workstations to perform different tasks. When it is planned to assemble several models of one product in the same robotic assembly line, the minimization of energy consumption becomes more difficult since the best assignment of tasks and robots to workstations must be found, taking into consideration all models. The authors cannot find in the literature a work that aims to minimize energy consumption in robotic assembly lines that produce several models with one configuration. Furthermore, the introduction of the heterogeneity of models and robots makes the problem more complex and hard, even for small-scale instances, and for this reason, the authors propose in this paper a Memory-Based Cuckoo Search Algorithm (MBCSA) to tackle this problem. The principle of memory is used in this new Cuckoo Search Algorithm in order to escape from the local optima and discover new search zones. Six problems of different sizes are generated and solved by the proposed MBCSA, and to evaluate its performance, two comparisons are made with two meta-heuristics, the genetic algorithm and another version of the cuckoo search algorithm. Obtained results show that this new version of the Cuckoo search algorithm is promising and can obtain good solutions for problems of different sizes.

## 1. Introduction

In manufacturing systems, assembly lines are known as the vital elements responsible for producing different products such as electronics, furniture, automobiles, and so on. Henry Ford was the first to introduce an assembly line system for mass production in his automobile factories. Since that time, several studies have been done aiming at developing these complex systems (Abdullah Make et al., 2017). In the assembly line, a set of workstations are linked together using a system that can move products from a workstation to another one, such as the conveyor belt, as shown in Fig. 1. To get the final product, a set of operations or tasks are performed in each workstation (Saif et al., 2014).

The design of an assembly line is a complex problem and requires different inputs such as the measurement criteria to be minimized or maximized; the precedence relations between tasks; task times, number of models to be produced in the line; and other constraints that may occur while designing the line. This problem is known in the literature as the assembly line balancing problem (ALBP) (Sivasankaran and Shahabudeen, 2014). The ALBP is classified into two versions, the simple version (SALBP) and the generalized version (GALBP). The difference between the two versions is that some assumptions of the SALBP can be neglected in the GALBP. For example, in the SALBP it is assumed that the line is designed for producing one single product model, whereas in the GALBP different models of one product can be produced on the same line (Becker and Scholl, 2006). Each version includes different types of problems, and each type is determined generally by the main objective of the problem; for instance, SALBP type-1 aims to minimize the number of workstations for a given cycle time, and SALBP type-2 aims to minimize the cycle time for a fixed number of workstations. For the generalized versions, the GALBP type-1 and the GALBP type-2 have the same aims as the SALBP-1 and the SALBP-2, respectively, but with different assumptions (Saif et al., 2014). The nature of the ALBP is NP-hard and it has been widely studied in the literature (Álvarez-Miranda and Pereira, 2019). When addressing new issues such as the mixture of models, zoning constraints, and the assignment of robots, the ALBP becomes more complicated (Xu and Xiao, 2008). The computing time increases due to several factors, such as the number of tasks, the number of operators (humans/robots), the existence of new constraints, and so on Bhattacharjee and Sahu (1990).

Due to the variation of customers' demands for different models of the same product, especially nowadays, manufacturers use special lines known as Mixed-Model Assembly Lines (MiMAL) to meet demands on time. Unlike traditional assembly lines, MiMALs generate several models in varying quantities and in an intermixed sequence

* Corresponding author.
*E-mail address:* lakhdar.belkharroubi@univ-mascara.dz (L. Belkharroubi).

https://doi.org/10.1016/j.engappai.2022.105112
Received 28 March 2022; Received in revised form 15 June 2022; Accepted 20 June 2022
Available online 1 July 2022

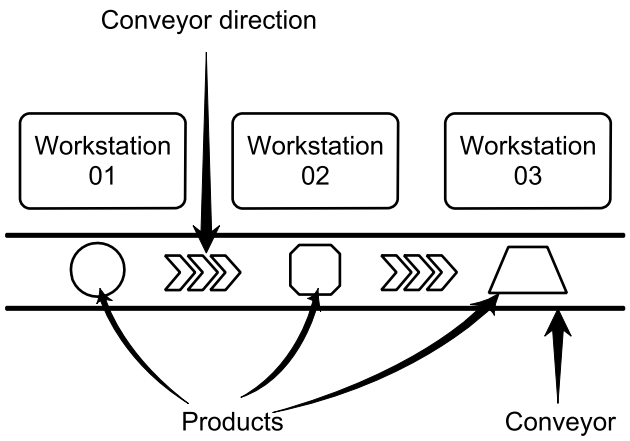



**Fig. 1.** The assembly line structure.

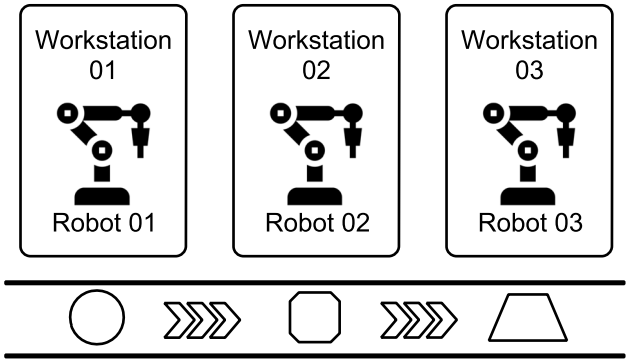



**Fig. 2.** Robotic assembly line.

(Çil et al., 2017a). The Mixed-Model Assembly Line Balancing Problem (MiMALBP) is the problem of assigning tasks to workstations in the MiMAL, which takes into account all models to be produced on the line. Assembly tasks could be performed manually by humans, automatically by robots, or by humans and robots in collaboration. To maintain competitiveness in the market and increase the flexibility of the assembly line, manufacturers use robots and other types of automated equipment in their lines instead of humans (Chutima, 2022). The assembly line in which tasks are performed by robots is known as the robotic assembly line (RAL) (see Fig. 2). Robotic technology is an essential part of Industry 4.0 that aims to create smart manufacturing systems (Bakar et al., 2019).

The integration of robots into the MiMAL created a new line version known as the Robotic Mixed-Model Assembly Line (RMiMAL) and another ALBP version known as the Robotic Mixed-Model Assembly Line Balancing Problem (RMiMALBP). The RMiMALBP seeks the best assignment of tasks and robots to a set of workstations in order to minimize or maximize various criteria measures such as cycle time, workstation count, and line efficiency (Chutima, 2022). As mentioned before, the main objective of the ALBP determines the problem type, and for the robotic assembly line balancing problem, several types have been solved, like the RALBP-1 that aims to minimize the number of workstations for a given cycle time, the RALBP-2 that aims to minimize the cycle time for a fixed number of workstations, the RALBP-E that aims to maximize the line efficiency by searching for the optimal values of the cycle time and the number of workstations, and the RALBP type cost that aims, for example, to minimize robot cost or robot setup cost (Rabbani et al., 2016; Nilakantan et al., 2017a). The same for the RMiMALBP problem; several types have been solved in order to optimize different objective values, taking into consideration all problem models with different assumptions and constraints (Chutima, 2022).

Energy is an important resource in manufacturing systems and the utilization of robots in assembly lines has led to increased energy consumption, and due to climate change, manufacturers are more interested in reducing energy consumption in order to create eco-friendly manufacturing systems (Sun et al., 2020; Mukund Nilakantan et al., 2015a). In the literature, researchers can find only some work aiming to minimize energy consumption in robotic assembly lines. The differences between done work are determined by the characteristics of the problem, including the line shape (straight line, U-shaped line, parallel line, and two-sided line), the resources to be optimized, such as the number of workstations, the cycle time, and consumed energy. In this paper, the authors focus on the minimization of the energy consumed in assembly lines that produce several models of one product and where assembly tasks are performed by robots. This problem can be divided into two stages: finding the best assignment of assembly tasks into workstations and finding the best assignment of robots with the least amount of energy consumption. When solving these two sub-problems, all models must be taken into consideration in order to find one configuration that satisfies all models' requirements. This is a very complex problem that has never been solved; all existing work on energy consumption minimization focuses solely on a single model of robotic assembly lines. As a solution to this problem, we propose in this work a new Cuckoo Search Algorithm version based on the concept of memory.

The remainder of this paper is as follows: Section 2 presents a literature review. Section 3 describes the problem and presents its mathematical formulation. In Section 4, the basic Cuckoo Search Algorithm is discussed. Section 5 describes the proposed memory-based Cuckoo Search Algorithm. In Section 6, a numerical example is proposed and solved using the MBCSA. Section 7 presents the computational results and their discussion. Finally, Section 8 concludes this work.

## 2. Literature review

The robotic assembly line balancing problem was introduced for the first time in 1991 by Rubinovits and Buckchin Rubinovitz and Bukchin (1991). The aim of the problem was to minimize the number of workstations by allocating the best available robot to each workstation to perform the assigned tasks. Other research has been done aiming to solve different versions of the robotic assembly line balancing problem using different methods like exact methods, heuristics, and meta-heuristics. The literature review in this paper consists of the robotic assembly line balancing problem, the RMiMALBP, and the minimization of energy consumption in the robotic assembly line.

Li et al. (2018a) proposed a discrete cuckoo search algorithm for solving the two-sided robotic assembly line balancing problem with the aim of minimizing the cycle time of the assembly line. Since the problem has two subproblems, namely assembly line balancing and robot allocation, the cuckoos were divided into two sub-swarms, each sub-swarm addressing a sub-problem. Nilakantan et al. (2017a) proposed a multi-objective co-operative co-evolutionary algorithm for minimizing the carbon footprint caused by the total energy consumption, and maximizing the line efficiency in robotic assembly line systems. Li et al. (2018b) developed mathematical models and simulated annealing algorithms to solve the robotic assembly line balancing problem type-2 in order to minimize the cycle time. Mathematical models have been implemented in the CLPEX solver to find optimal solutions for small-sized problems, and to tackle large-sized problems; two simulated annealing algorithms, restarted SA and original SA, are proposed. Çil et al. (2017a) presented a mathematical model to solve the robotic parallel assembly line balancing problem type-2 (RPALBP-2) to minimize the cycle time, and due to the NP-hardness of the problem, they have proposed three meta-heuristics to solve large-sized problems, iterative beam search (IBS), best search method based on IBS (BIBS), and cutting BIBS (CBIBS).

Nilakantan et al. (2017b) addressed the robotic assembly line balancing problem with two objectives: minimizing the cycle time and minimizing the total assembly line cost. They developed two models, a time-based model to optimize cycle time and a cost-based model to optimize assembly line cost. A Differential evolution (DE) algorithm was used to solve the RALBP due to its complexity. Yadav and Agrawal (2022) addressed the robotic two-sided assembly line balancing problem (RTALBP) with zoning constraints; the objective was to maximize the workload on each workstation in order to minimize the number of workstations for a fixed cycle time. To solve the RTALBP, they developed the branch and bound algorithm using the Lingo solver.

Rabbani et al. (2016) proposed two multi-objective meta-heuristics for solving the RMiMALBP with the aim of minimizing the cycle time as a primary objective. The first algorithm is a non-dominated sorting genetic algorithm (NSGA-2) and the second one is multi-objective particle swarm optimization (MOPSO). Aghajani et al. (2014) presented a mixed integer programming model for the RMiMALBP with setup times in order to minimize the cycle time for a given number of mated workstations. They have proposed a simulated annealing (SA) algorithm as a meta-heuristic method to solve the problem. Çil et al. (2017b) proposed mixed-integer linear programming models and an efficient heuristic algorithm based on the beam search method in order to minimize the sum of cycle times over all models in RMIM lines.

Zhang et al. (2019) addressed the energy-efficient U-shaped robotic assembly line balancing problems. They developed a non-linear multi-objective mixed-integer program in order to model the problem, and to solve it, they used a Pareto artificial bee colony algorithm (PABCA). Nilakantan et al. (2015) proposed a particle swarm optimization algorithm in order to minimize energy consumption in a U-shaped robotic assembly line where the number of workstations is fixed. Li et al. (2016) proposed a restarted simulated annealing algorithm to solve the two-sided robotic assembly line with two objectives, minimizing energy consumption and minimizing cycle time. Nilakantan et al. (2017b) have proposed a differential evolution (DE) in order to minimize the energy consumption in a straight robotic assembly line and thereby help to reduce energy costs. Sun et al. (2020) addressed the energy-efficient robotic assembly line balancing problem with the criteria to minimize the cycle time and total energy consumption. The authors proposed a multi-objective algorithm (EDA) and a bound-guided hybrid estimation of distribution algorithm in order to solve the problem. Mukund Nilakantan et al. (2015a) proposed two models with a dual focus on time and energy for solving the robotic assembly line. The time-based model is used to minimize the cycle time and the energy-based model is used to minimize the energy consumption. A particle swarm optimization was used to solve the problem.

To the best knowledge of the authors of this paper, there are not enough papers that deal with the minimization of the energy consumed in robotic assembly lines, and in the existing ones, there is no paper that deals with the problem of minimizing the energy consumed in robotic assembly lines that assemble several models as illustrated in Table 1. In this paper, we propose a new version of a bio-inspired algorithm known as the Cuckoo Search Algorithm to solve the problem. There are two types of meta-heuristics: memory-based meta-heuristics, those who use the memory technique while solving optimization problems, and memory-less meta-heuristics, those who are not based on the memory technique (Agrawal and Kaur, 2017). Our new CSA version is a memory-based meta-heuristic, so we can call it Memory-Based Cuckoo Search Algorithm (MBCSA).

The memory concept is one of the powerful techniques that can be used in meta-heuristics in order to get a compromise between diversification and intensification in the search space (Li et al., 2012). In other words, it helps the algorithm to discover new search zones by neglecting already discovered zones that are placed in the memory, and as a result, new solutions can be found. Several memory-based meta-heuristics have been proposed to solve different optimization problems. A memory-based Greywolf optimizer was proposed for the global optimization tasks (Gupta and Deep, 2020). To solve the economic dispatch problem in micro-grids, a memory-based gravitational search algorithm (MBGSA) was proposed (Younes et al., 2021). A multistart adaptive memory-based tabu search algorithm was proposed to solve the heterogeneous fixed-fleet open vehicle routing problem (Li et al., 2012). A memory-based iterated local search algorithm was proposed to solve the multi-depot open vehicle routing problem (Brandão, 2020). Memory based Hybrid Dragonfly Algorithm for numerical optimization problems (Sree Ranjini and Murugan, 2017). The Memory-Based NSGA-II Algorithm for Dynamic Multi-objective Optimization Problems (Sahmoud and Topcuoglu, 2016). Adaptive memory-based local search for the maximum satisfiability problem (Lü and Hao, 2012).

## 3. Energy-efficient RMiMALBP

### 3.1. Problem description and assumptions

In the same RMiML, several models are planned to be produced in an intermixed sequence. Each model has its own precedence relations diagram, and all diagrams of models can be combined into one diagram. A set of workstations are arranged in the line, and in each workstation, a set of tasks are assigned based on the combined precedence relations diagram. From a set of available robots, the robot with the minimum energy consumption is chosen to be allocated to the workstations in order to minimize the total energy consumed by the line for a specific sequence of different models. The task time of a specific task varies from one model to another, which means that the energy consumed by a specific robot varies as well from one model to another. The energy consumed in each workstation includes both the energy consumed by the robot while performing tasks and the energy consumed by the robot while in standby mode. Each robot has its own characteristics, including the processing time and the energy needed to accomplish a specific task. For each robot, the processing time and the energy consumed vary from one model to another. The Energy-Efficient RMiMAALBP is more complex than all existing problems related to the minimization of energy consumed due to the heterogeneity of robots and the heterogeneity of models. For each obtained sequence of tasks, the best assignment of robots with minimal energy consumption must be found, taking into consideration all models. This complexity is considered a big challenge when searching for a satisfying solution since the search space becomes larger when the heterogeneity of models and the heterogeneity of robots are combined at the same time (see Fig. 3).

The objective is to minimize the total energy consumed by robots in the assembly line by finding the best configuration (assignment of tasks and robots) that satisfies all model requirements. As shown in the illustrative example (Fig. 4), the assembly line is designed to assemble two different models, and there are two different robots assigned to two workstations to perform a set of tasks according to the combined diagram. This configuration may not be the best one in terms of the energy consumed.

The assumptions listed below are considered in the model formulation:

1. The line is straight and several models are assembled in an intermixed sequence.
2. A task cannot be assigned to different workstations.
3. The assignment of tasks is constrained by precedence relations.
4. The task time depends on the assigned robot.
5. There are several types of robots, and there is no limit for each type of robot.
6. The same type of robot can be assigned to different workstations.
7. Only one robot can be assigned to a workstation.
8. Tasks can be performed by any type of robot.
9. The energy consumed by a robot varies from one model to another.
10. Based on the energy consumed by robots to produce each model, the average energy consumed can be calculated and used while optimizing the line.

**Table 1**
Papers that dealt with the minimization of the energy consumed.

| Paper | Objective | Models | Solution |
|---|---|---|---|
| Zhang et al. (2019) | Min(energy consumed) | Single | PABCA |
| Nilakantan et al. (2015) | Min(energy consumed) | Single | PSO |
| Li et al. (2016) | Min(energy consumed) and Min(cycle time) | Single | RSA |
| Janardhanan et al. (2018) | Min(energy consumed) | Single | DEA |
| Sun et al. (2020) | Min(energy consumed) and Min(cycle time) | Single | BGHEDA |
| Mukund Nilakantan et al. (2015a) | Min(energy consumed) and Min(cycle time) | Single | PSO |

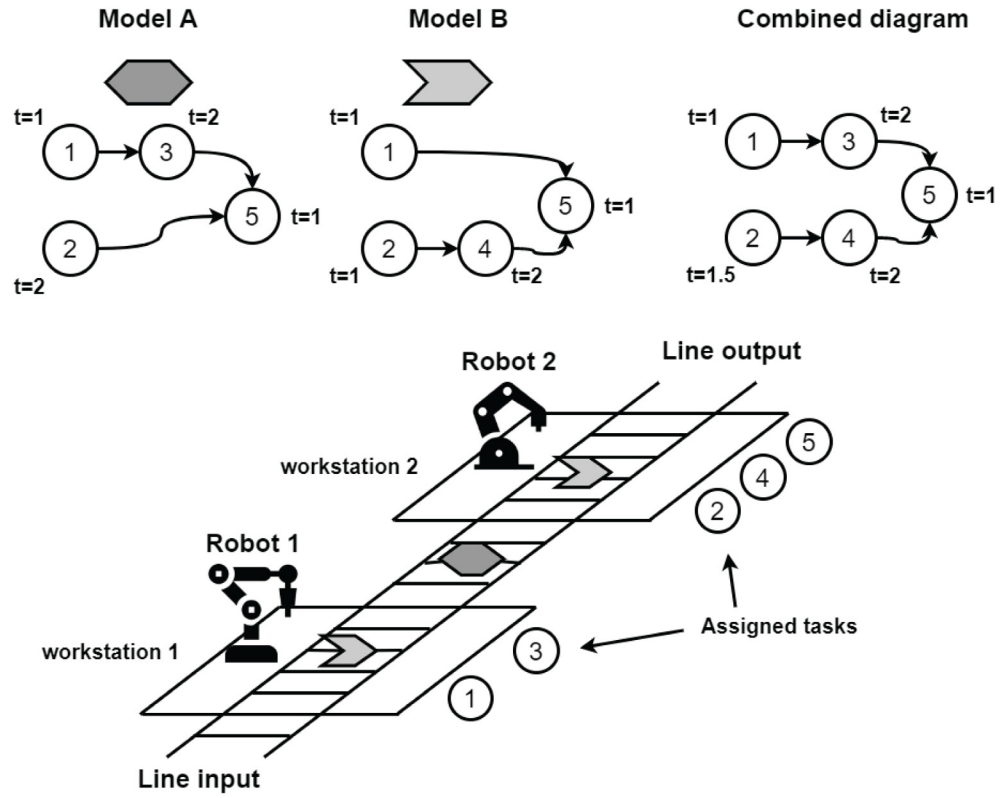



**Fig. 3.** Illustrative example.

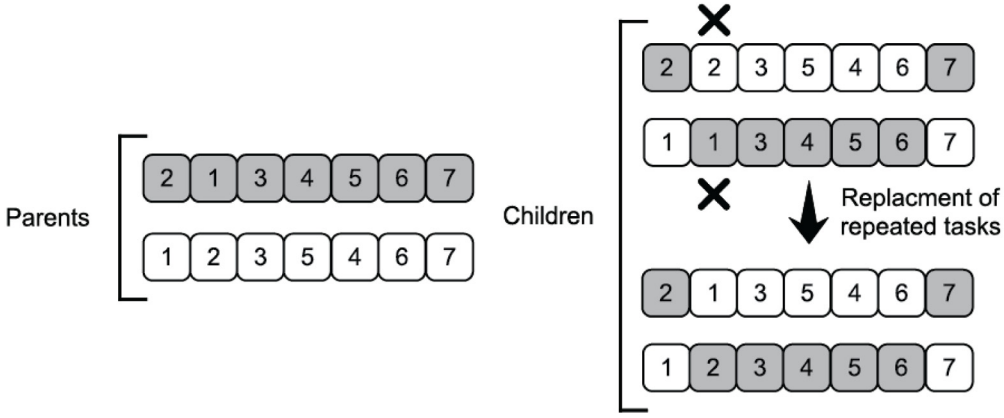



**Fig. 4.** Generation of new solutions using two-point crossover.

### 3.2. Notations

| | |
|---|---|
| Parameters: | |
| $N$ | Number of tasks. |
| $i$ | Task index. |
| $W$ | Number of workstations. |
| $k$ | Workstation index. |
| $r$ | Robot index. |
| $t_{ri}$ | Processing time needed by robot $r$ to complete task $i$. |
| $e_{ri}$ | Energy consumed by robot $r$ to perform task $i$. |
| $es_r$ | Energy consumed each time unit by robot $r$ in the standby period. |
| $P_i$ | Immediate predecessors of task $i$ |
| $ECP_{rk}$ | Energy consumed by robot $r$ to perform assigned tasks in workstation $k$. |
| $ECS_{rk}$ | Energy consumed by robot $r$ during the standby period in workstation $k$. |
| Decision variables: | |
| $X_{ik}$ | Equal to 1 if task $i$ is assigned to workstation $k$, 0 otherwise |
| $Y_{rk}$ | Equal to 1 if robot $r$ is assigned to workstation $k$, 0 otherwise |
| $TEC$ | Total energy consumed |
| $EC_k$ | Energy consumed by workstation k |
| $CT$ | The cycle time of obtained assignment (or maximum workstation time). |

### 3.3. Mathematical formulation

The proposed mathematical model aims to minimize total energy consumption in the RMiMAL. By combining all precedence relations diagrams into one diagram, we can ameliorate models that have been used to solve robotic assembly line balancing problems with one single model. Our model is based on the energy-based model that is proposed in Mukund Nilakantan et al. (2015a), This model focuses on minimizing the total energy consumption of robotic assembly lines.

$$min \quad TEC = \sum_{k=1}^{W} EC_k \tag{1}$$

Subject to:

$$EC_K = ECP_{rk} + ECS_{rk} \tag{2}$$

$$ECP_{rk} = \sum_{i=1}^{N} e_{ri} * X_{ik} \leq for \quad k = 1, 2, \ldots, W \tag{3}$$

$$ECS_{rk} = (CT - \sum_{i=1}^{N} t_{ri} * X_{ik}) * es_r \tag{4}$$

$$\sum_{k=1}^{W} k * X_{hk} \leq \sum_{k=1}^{W} k * X_{ik} \quad whereh \in P_i \tag{5}$$

$$\sum_{k=1}^{W} X_{ik} = 1 \quad for \quad i = 1, 2, \ldots, N \tag{6}$$

$$\sum_{k=1}^{W} Y_{rk} = 1 \quad for \quad i = 1, 2, \ldots, N \tag{7}$$

$$X_{ik} \in \{0, 1\} \tag{8}$$

$$Y_{ik} \in \{0, 1\} \tag{9}$$

The objective function (1) minimizes total energy consumption. Eq. (2) calculates the energy consumption of a specific workstation. Eq. (3) calculates the energy consumed by robot r to perform assigned tasks at workstation $k$. Eq. (4) calculates the energy consumed by robot r during the standby mode at workstation $k$. Eq. (5) ensures that all precedence relations among tasks are respected. Eq. (6) ensures that each task is assigned to only one workstation. Eq. (7) ensures that only one robot is assigned to one workstation. Eqs. (8) and (9) indicate the types of decision variables.

## 4. Cuckoo search algorithm

The Cuckoo search algorithm (CSA) is a nature-inspired meta-heuristic introduced by Xin-She Yang and Suash Deb in 2009 (Yang and Deb, 2009). The concept of this algorithm is based on the brood parasitism of some cuckoo species, such as Ani and Guira cuckoos. These cuckoos lay their eggs in the nests of other host birds, and sometimes they also throw the eggs of the host birds to increase the hatching chances of their own eggs (Du and Swamy, 2016). The process of laying eggs is combined with the levy flight behavior of some birds and fruit flies (Xin-She, 2014). Cuckoo eggs can be discovered by the host birds; in this case, the host birds will either discard the discovered eggs or abandon their nest and build new nests elsewhere (Yang and Deb, 2009).

In the CSA, the host bird egg presents a solution, the cuckoo egg presents a new solution, and the goal is to replace the old solution (host bird egg) with the new solution (cuckoo egg). The CSA follows three rules: each cuckoo lays one egg at a time in a randomly chosen nest, the best nests that contain high-quality eggs will be passed to the next generations, the number of available host nests is fixed; and the probability of discovering laid eggs by cuckoos is $P_a$ where $P_a \in [0, 1]$ (Du and Swamy, 2016; Xin-She, 2014).

**Algorithm 1** Cuckoo Search Algorithm

```
Objective function: f(X), X = (X1, X2, ...Xd)
Generate Initial population of N host nests
while termination criterion is not met do
    Generate a cuckoo (say, Xi) randomly using lévy flights
    Choose a nest among N (say, Xj) randomly
    if f(Xi) < f(Xj) then
        Replace Xj by Xi in the population
    end if
    A fraction Pa of worst nests are abandoned and new ones are built
    Keep the best solutions/nests
    Rank the solutions/nests based on their fitness and find the best current solution
end while
```

## 5. A memory based cuckoo search algorithm for the EERMiMALB problem

In order to solve the energy-efficient RMiMALBP, we propose a memory based Cuckoo Search Algorithm (MBCSA) with some modifications in some steps. Two works that have dealt with the RALBP using the CSA can be found in the literature, a discrete cuckoo search algorithm for solving the two-sided RALBP with the objective of minimizing the cycle time (Li et al., 2018a), and a Hybrid Cuckoo Search Particle Swarm Optimization Algorithm was proposed to solve the RALBP with the objective of minimizing the cycle time (Mukund Nilakantan et al., 2015b). This algorithm is more intelligent in comparison with exciting CSA versions, and it is a new contribution in the field of artificial intelligence and computer science like recently published works such as Roy (2022) and Kapgate (2022). The proposed MBCSA in this paper consists of the following steps.

**Algorithm 2** A memory-based Cuckoo Search Algorithm

```
Ojective funciton: f(X), X = (X1, X2, ...Xd)
Generate Initial population of N host nests
Initialize the memory M, max_search, max_generations
while max_generations do
    while max_search do
        Select randomly a nest (say, Y) from the population
        if Y is not in M then
            Reset max_search to the initial value
            Exit from while loop
        end if
        Decrement max_search
        if max_search is equal to 0 then
            End the algorithm
        end if
    end while
    Generate a list of neighbors of Y (say, NL(Y)) using the swap mutation with repair mechanism
    Stock the nest Y and all its neighbors NL(Y) in M
    Select the best neighbor from NL(Y) as the new cuckoo (say, C)
    Choose a nest among N (say, H) randomly
    if f(C) < f(H) then
        Replace H by C in the population
    end if
    A fraction Pa of worst nests are abandoned and new ones are built
    while max_search do
        Choose two parents from the reminder of the population
        Apply the crossover on selected parents with the repair mechanism
        if Generated solution are not in M then
            Reset max_search to the initial value
            Exit from while loop
        end if
        Decrement max_search
        if max_search is equal to 0 then
            End the algorithm
        end if
    end while
    Replace abandoned solutions by generated one
    Rank the solutions based on the objective value
    Select the best solution as the best current solution
    Decrement max_generations
end while
```

### 5.1. Initialization of the first population

The population presents a set of nests (solutions), and in order to start the MBCSA, the first population is generated randomly. Each solution presents a specific assignment of tasks and robots into workstations. The assignment of tasks must be feasible according to the precedence relations between tasks. After obtaining a feasible assignment, the best robots with the lowest energy consumption are assigned to workstations. Generated solutions are not necessarily optimal.

### 5.2. Generation of new cuckoo solution and memory usage

To generate a new cuckoo solution, a nest (say X) is chosen randomly from the population, and the list of neighbors of X (say N(X)) is generated. The process of choosing a random nest (X) is repeated for a number fixed by the programmer until a nest that does not exist in the memory is obtained. If the counter reaches the number of tries, then, the algorithm stops. A neighbor is obtained by applying a swap mutation on X by changing the positions of two tasks chosen randomly, and to reach the number of neighbors that is fixed by the programmer, the swap mutation is repeated several times. Each solution in N(X) is unique and has its own objective value. Neighbors are ranked based on their objective values, and the best one is selected as the new cuckoo solution.

The new cuckoo solution was evaluated and compared with another solution randomly selected (say Y) from the current population. In our case, if the new cuckoo has minimal energy consumption in comparison with Y, the MBCSA replaces Y with the new cuckoo in the population. Finally, selected host X and its neighbors in N(X) are placed in memory. The reason behind this technique is to discover new solutions in future iterations by neglecting any previously obtained solution that exists in memory. Furthermore, the algorithm will avoid searching for solutions that are already obtained in previous solutions to minimize CPU time, especially when search space is important.

### 5.3. Replacement of abandoned solutions

The algorithm abandons a fraction $P_\alpha$ of the worse solutions discovered by the cuckoo, and to replace abandoned solutions, a new technique is used in the proposed MBCSA. The crossover operator is applied to two nests (parents) selected randomly from the remaining better solutions to generate new solutions. For instance, if two solutions are abandoned, the crossover operator replaces them with the two newly generated solutions; if more than two solutions are abandoned, the crossover process is repeated on other selected solutions that present a different couple until the number of individuals is reached to fill the gap in the population.

The reason behind choosing new couples every time is to obtain new and different solutions. After applying the crossover operation, some tasks can be repeated in generated solutions, and to tackle this problem, the algorithm verifies which tasks are repeated to replace them with missing ones. At this stage, the algorithm verifies if the generated solutions after applying the crossover do not exist in memory. In the case where solutions do not exist in the memory, the MBCSA pushes them into the population; otherwise, the crossover operator is repeated until new solutions that do not exist in the memory are found.

### 5.4. Repair mechanism

In order to maintain the feasibility of solutions, the repair mechanism is used in the MBCSA. After applying the swap mutation and the crossover operator, the repair mechanism verifies the feasibility of solutions by checking if the order of tasks respects the precedence relations between tasks. If the new solution is not feasible, the positions of tasks that make the solution unfeasible are changed. As shown in Fig. 5, the obtained solution does not respect the precedence relations shown in the diagram; thus, the repair mechanism is applied by swapping tasks 5 and 2 the first time, and tasks 5 and 3 the second time.

### 5.5. Fitness evaluation

In order to evaluate obtained solutions, the energy-based model proposed in Nilakantan et al. (2015) is used in MBCSA to determine the objective value (total energy consumed) of solutions. In the energy-based model, tasks are assigned to workstations based on a specific energy consumption value, and robots with the lowest energy consumption are selected to be assigned to workstations. The assignment process starts with an initial energy consumption value of $E_0$ calculated using Eq. (10), and if all tasks cannot be assigned with this value, then, it is incremented by a small value fixed by the programmer. The assignment process is repeated until all tasks of the corresponding solution/sequence are assigned using the best robot-to-workstation assignment.

$$E_0 = [\sum_{i=1}^{N} min(e_{ri})/N_w] \tag{10}$$

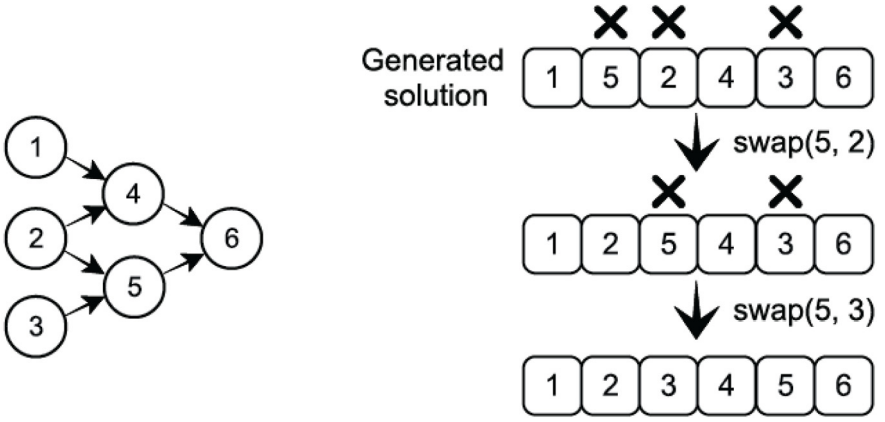



**Fig. 5.** Repair mechanism.

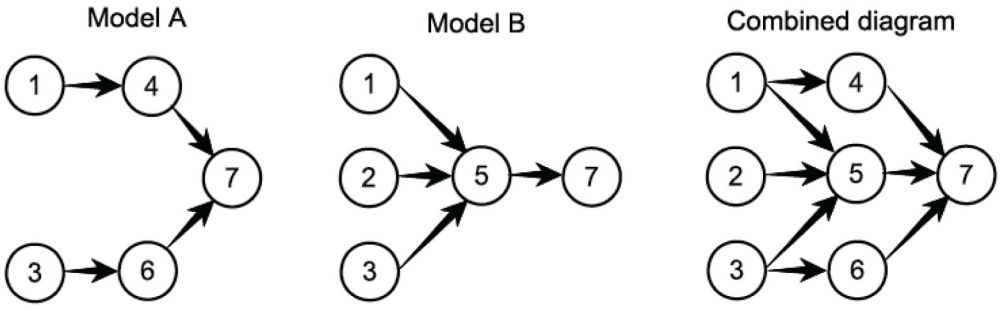



**Fig. 6.** Precedence relations diagrams of the numerical example.

**Table 2**
Model A.

| Task | Robot 1 | | Robot 2 | |
|---|---|---|---|---|
| | Task time | Energy | Task time | Energy (kJ) |
| 1 | 1 | 1 | 1 | 1 |
| 3 | 2 | 2 | 1 | 1 |
| 4 | 1 | 1 | 1 | 1 |
| 6 | 1 | 1 | 2 | 2 |
| 7 | 1 | 1 | 1 | 1 |

where $e_{ri}$ is the energy consumed by robot $r$ $(r = 1, \ldots, N_r)$ to perform task $i$ and $N_w$ is the number of workstations.

The energy consumption in each workstation is equal to the sum of the amount of energy consumed while the assigned robot performs assigned tasks and the energy consumed during the standby period (when the robot does not perform any tasks).

## 6. Numerical example

In this section, an example with two models (A and B) and two robots (R1 and R2) is solved by the proposed cuckoo search algorithm. Fig. 6 shows the diagrams of models A and B and the combined diagram. It is assumed in this example that both robots consume 1 kJ per one-time unit during the processing period, while in the standby period, robot 1 and robot 2 consume, respectively, 0.5 kJ and 0.4 kJ per one-time unit. The processing time taken by each robot to accomplish a specific task for each model and the corresponding energy consumed are illustrated in Tables 2 and 3.

**Table 3**
Model B.

| Task | Robot 1 | | Robot 2 | |
|---|---|---|---|---|
| | Task time | Energy | Task time | Energy (kJ) |
| 1 | 1 | 1 | 1 | 1 |
| 2 | 1 | 1 | 2 | 2 |
| 3 | 2 | 2 | 2 | 2 |
| 5 | 2 | 2 | 1 | 1 |
| 7 | 1 | 1 | 1 | 1 |

**Table 4**
Combined model A and model B.

| Task | Robot 1 | | Robot 2 | |
|---|---|---|---|---|
| | Task time | Energy | Task time | Energy (kJ) |
| 1 | 1 | 1 | 1 | 1 |
| 2 | 1 | 1 | 2 | 2 |
| 3 | 2 | 2 | 1.5 | 1.5 |
| 4 | 1 | 1 | 1 | 1 |
| 5 | 2 | 2 | 1 | 1 |
| 6 | 1 | 1 | 1 | 1 |
| 7 | 1 | 1 | 1 | 1 |

In order to solve the problem, taking into consideration the mixture of models in the line, we combine Tables 1 and 2 into Table 3 to make the problem easier and to obtain results that satisfy all models. The average of the task time and the energy consumed for each common task are calculated as shown in Table 4. Table 5 shows the used parameters to solve the numerical example. The max search parameter presents the number of times in which the algorithm tries to find a random nest that does not exist in the memory.

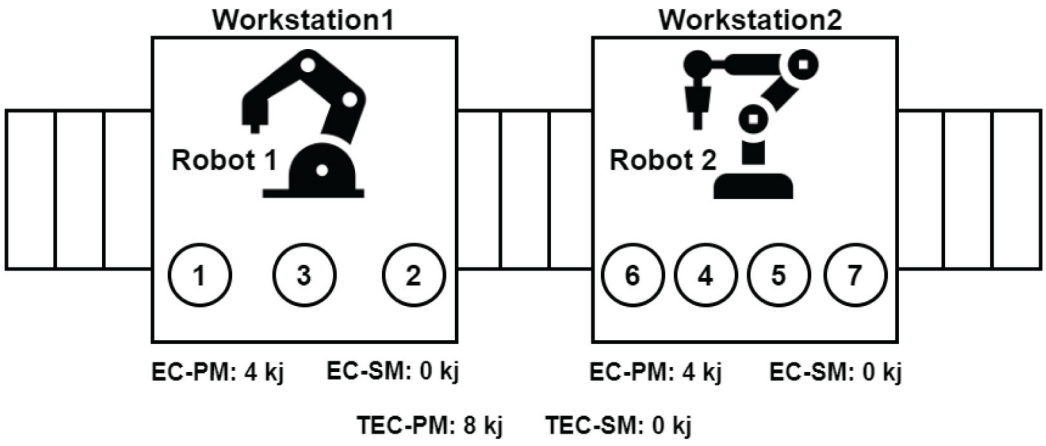



**Fig. 7.** The assignment of tasks and robots to workstations.

**Table 5**
Used parameters to solve the numerical example.

| Parameter | Value |
|---|---|
| Maximum number of generations | 100 |
| Population size | 10 |
| Fraction $P_a$ | 0.15 |
| max_search | 100 |

**Table 6**
The first and the final populations for the numerical example.

| Nests | First population | | | Final Population | | |
|---|---|---|---|---|---|---|
| | TEC-PM | TEC-SM | TEC | TEC-PM | TEC-SM | TEC |
| 1 | 8 | 0 | 8 | 8 | 0 | 8 |
| 2 | 8 | 0 | 8 | 8 | 0 | 8 |
| 3 | 7.5 | 0.6 | 8.1 | 8 | 0 | 8 |
| 4 | 8.5 | 0.2 | 8.7 | 8 | 0 | 8 |
| 5 | 8.5 | 0.75 | 9.25 | 8 | 0 | 8 |
| 6 | 8 | 0 | 8 | 8 | 0 | 8 |
| 7 | 8.5 | 0.2 | 8.7 | 8 | 0 | 8 |
| 8 | 8.5 | 0.95 | 9.45 | 8 | 0 | 8 |
| 9 | 8.5 | 0.95 | 9.45 | 8 | 0 | 8 |
| 10 | 8.5 | 0.95 | 9.45 | 8 | 0 | 8 |

Table 6 shows the initial and final populations. There are 10 solutions (nests) in the population, and each individual (nest) presents the assignments of tasks and robots to workstations. The total energy consumed during the processing mode (TEC-PM) and the standby mode (TEC-SM) for each solution is also illustrated in the table. The total energy consumed (TEC), which presents the objective value and is equal to the sum of TEC-PM and TEC-SM, is also calculated.

Based on the objective value, the reader can observe that in the first population, there are bad and good solutions. For example, there are three individuals with an objective value equal to 8, one individual with an objective value equal to 8.1, two individuals with an objective value equal to 8.7, and finally four individuals with an objective value equal to 9.25. After 100 generations, we can observe that all nests in the final population present good solutions, having the same objective value of 8.

By observing the obtained results for the proposed numerical example, we can conclude that the minimal energy consumed that can be obtained is 8, and the best assignment of robots that gives this value is when robots 1 and 2 are assigned to workstations 1 and 2, respectively.

**Table 7**
Problems' specifications.

| Problem | Tasks | Models | Robots | Workstations |
|---|---|---|---|---|
| P1 | 12 | A, B | 1, 2 | 3 |
| P2 | 20 | A, B | 1, 2 | 4 |
| P3 | 25 | A, B, C | 1, 2, 3 | 5 |
| P4 | 40 | A, B | 1, 2, 3 | 7 |
| P5 | 57 | A, B | 1, 2, 3, 4 | 10 |
| P6 | 72 | A, B, C | 1, 2, 3 | 12 |

Fig. 7 shows one solution selected from the final population. Robot 1 performs tasks 1, 2 and 3 in workstation 1, while robot 2 performs tasks 6, 4, 5, and 7 in workstation 2.

## 7. Computational results and discussion

The proposed MBCSA was implemented using Python programming language on a personal computer with a CPU Intel Dual-core 1.7 GHz, and 8 GB of memory. To test the performance of the algorithm, six problems of different sizes were generated during this study and are accessible at https://github.com/Belkharroubi-Lakhdar/Robotic-Mixed-Model-AL.git. As shown in Table 7 each problem has its own information, including the precedence relations diagram, the assembly tasks, the number of models, the number of robots, and the number of workstations. In problem 1, there are 12 tasks and two models (A and B), two robots (1 and 2), and three workstations. Problem 2 has 20 tasks, two models (A and B), two robots (1 and 2), and four workstations. In problem 3, there are 25 tasks, three models (A, B, and C), three robots (1, 2 and 3), and five workstations. In problem 4, there are 40 tasks, two models (A and B), three robots (1, 2 and 3), and seven workstations. In problem 5, there are 57 tasks, two models (A and B), four robots (1, 2, 3, and 4), and 10 workstations. Finally, in problem 6, there are 72 tasks, three models (A, B, and C), and 12 workstations.

In order to test the performance of the proposed MBCSA, two comparisons are made in this work. In the first comparison, the MBCSA was compared with the genetic algorithm, and each problem was solved nine times using both algorithms. In the second comparison, the MBCSA was compared with another memory less CSA (MLCSA) similar to the discrete CSA that was used in Li et al. (2018a) to solve the two-sided robotic assembly line balancing problem. The MLCSA was executed nine times for each problem. Table 8 present used parameters for solving each problem using the MBCSA and the MLCSA version. Used parameters in the GA are shown in Table 9. The obtained results

**Table 8**
Used parameters in the MBCSA and the MLCSA.

| Problems | Generations | Population's size | Fraction $P_a$ |
|---|---|---|---|
| P1 | 200 | 20 | 0.15 |
| P2 | 300 | 30 | 0.15 |
| P3 | 300 | 30 | 0.15 |
| P4 | 300 | 100 | 0.15 |
| P5 | 300 | 100 | 0.15 |
| P6 | 300 | 100 | 0.15 |

**Table 9**
Used parameters in the GA.

| Problems | Generations | Population's size | Crossover | Mutation | Elitism |
|---|---|---|---|---|---|
| P1 | 200 | 20 | 0.25 | 0.1 | 15% |
| P2 | 300 | 30 | 0.25 | 0.1 | 10% |
| P3 | 300 | 30 | 0.25 | 0.1 | 10% |
| P4 | 300 | 100 | 0.25 | 0.1 | 8% |
| P5 | 300 | 100 | 0.25 | 0.1 | 7% |
| P6 | 300 | 100 | 0.25 | 0.1 | 7% |

from the first comparison, including the objective value (total energy consumed), the cycle time, and the CPU time in seconds, are shown in Table 10. In Table 12, the best objective values obtained by the MLCSA are compared with those obtained by the MBCSA.

Starting with the obtained results for problem 1 using both the MBCSA and the GA 9 times, we can observe that both algorithms obtained the same objective value (16.4 kJ), and the same cycle time (5.5) in all executions. The only difference that can be remarked is that the MBCSA outperforms the GA in terms of CPU time in all executions. For problem 2, both algorithms obtained approximately the same objective values. The minimal obtained value is 24.3 kJ, and was obtained by both the MBCSA and the GA. Also, the MBCSA outperforms the GA in terms of CPU time in all executions. The average of the obtained objective values by the MBCSA is 24.69 kJ, while for the GA it is 24.58 kJ. For problem 3, also, both the MBCSA and the GA obtained approximately the same values, and the minimal obtained value is equal to 39.8 kJ, which was obtained by the MBCSA. The averages of the obtained values are very close at 41.19 kJ for the MBCSA and 41.27 kJ for the GA. In terms of CPU time, the MBCSA outperforms the GA.

For problem 4, the averages of obtained values by the MBCSA and the GA are respectively 75.71 kJ and 78.27 kJ, which means that the MBCSA obtained better values and this can be noted in the table. The minimal obtained value by the MBCSA is equal to 72.61 kJ. In terms of CPU time, the MBCSA outperforms the GA in all executions. For problem 5, the averages of the obtained values by the MBCSA and the GA are respectively 123.70 kJ and 125.17 kJ. The best obtained value by the MBCSA is equal to 121.39 kJ, compared to 123.19 kJ for the GA. In terms of CPU time, the GA outperforms the MBCSA in most cases. For problem 6, the GA outperforms the MBCSA in terms of CPU time, but in terms of objective values, the MBCSA is better, with an average equal to 164.76 kJ, while for the GA it is equal to 168.18 kJ. The best obtained values by the proposed MBCSA and the GA are respectively 159.95 kJ and 165.42 kJ. Obtained objective values for all problems are plotted in Fig. 8 in order to illustrate the difference between the MBCSA and the GA. We can easily conclude that the proposed MBCSA was better than the GA in obtaining minimal energy consumption for the large problems 4, 5, and 6. For problems 1, 2, and 3, both algorithms obtained almost the same values.

Table 11 shows the details of the best obtained solutions by both algorithms. The assignment of robots $R$ into workstations $W$, the processing time $PT$ and the idle time $ID$ n each workstation, the energy consumed in both processing mode $E(PM)$ and the standby mode $E(SM)$, and the total energy consumed $TE$.

The obtained objective values by both the MBCSA and MLCSA are presented in Table 12. Each problem was solved nine times. We can see from these results that the MBCSA beats the MLCSA when it comes to tackling large-scale problems, according to the best obtained values in bold text. These findings demonstrate the relevance of memory integration in assisting the algorithm in discovering good solutions that would be difficult to find using a memory-free approach.

**Table 10**
All executions of both algorithms for solving all problems.

| Problem | MBCSA | | | GA | | |
|---|---|---|---|---|---|---|
| | C | TEC | CPU time (s) | C | TEC | CPU time (s) |
| P1 | 5.5 | 16.4 | 1.54 | 5.5 | 16.4 | 10.19 |
| | 5.5 | 16.4 | 2.15 | 5.5 | 16.4 | 9.57 |
| | 5.5 | 16.4 | 1.81 | 5.5 | 16.4 | 9.32 |
| | 5.5 | 16.4 | 3.30 | 5.5 | 16.4 | 9.56 |
| | 5.5 | 16.4 | 3.00 | 5.5 | 16.4 | 9.75 |
| | 5.5 | 16.4 | 1.36 | 5.5 | 16.4 | 11.60 |
| | 5.5 | 16.4 | 2.28 | 5.5 | 16.4 | 12.48 |
| | 5.5 | 16.4 | 1.85 | 5.5 | 16.4 | 9.65 |
| | 5.5 | 16.4 | 2.03 | 5.5 | 16.4 | 9.18 |
| P2 | 7 | 24.65 | 21.83 | 7.5 | 24.55 | 70.15 |
| | 7 | 24.65 | 17.91 | 7 | 24.65 | 71.54 |
| | 7.5 | 24.3 | 23.18 | 7 | 24.65 | 68.21 |
| | 7.5 | 24.3 | 20.23 | 7.5 | 24.55 | 65.29 |
| | 8 | 24.79 | 13.19 | 7 | 24.35 | 67.96 |
| | 7.5 | 25.55 | 27.86 | 8.5 | 25.35 | 67.12 |
| | 7.5 | 24.8 | 19.99 | 7.5 | 24.3 | 65.23 |
| | 7 | 24.6 | 21.13 | 7.5 | 24.3 | 76.38 |
| | 7.5 | 24.55 | 19.50 | 7.5 | 24.55 | 67.42 |
| P3 | 8.5 | 41.5 | 29.55 | 8.5 | 41.5 | 71.07 |
| | 8.5 | 41.3 | 47.21 | 9 | 42.35 | 77.70 |
| | 8.5 | 41.5 | 62.97 | 8.5 | 41.5 | 73.24 |
| | 8 | 41.45 | 33.03 | 8.5 | 41.5 | 77.27 |
| | 8 | 40.2 | 27.94 | 8.5 | 41.35 | 77.68 |
| | 7.5 | 39.8 | 28.64 | 8 | 40.2 | 83.83 |
| | 8.5 | 41.5 | 28.43 | 8 | 40.0 | 79.36 |
| | 9.5 | 43.5 | 56.18 | 8 | 41.5 | 83.64 |
| | 8 | 40.0 | 35.9 | 8.5 | 41.5 | 77.30 |
| P4 | 7.7 | 75.73 | 254.35 | 8 | 78.06 | 313.27 |
| | 7 | 72.61 | 235.50 | 8 | 78.21 | 332.09 |
| | 7.5 | 75.75 | 171.87 | 8 | 78.15 | 334.41 |
| | 8 | 77.6 | 204.86 | 8 | 79.1 | 361.2 |
| | 7.5 | 75.4 | 421.30 | 8 | 78.61 | 355.45 |
| | 7.2 | 73.11 | 91.02 | 7.5 | 77.2 | 344.04 |
| | 7.5 | 74.45 | 239.74 | 8 | 78.75 | 325.42 |
| | 8 | 78.75 | 294.45 | 8 | 77.66 | 348.96 |
| | 8 | 78.01 | 182.86 | 8 | 78.71 | 351.52 |
| P5 | 9.2 | 125.20 | 790.58 | 9.1 | 126.48 | 996.63 |
| | 8.85 | 121.39 | 1618.92 | 8.85 | 124.45 | 996.66 |
| | 8.95 | 124.85 | 1706.37 | 8 | 126.97 | 951.62 |
| | 9.2 | 125.21 | 1448.19 | 9.25 | 126.55 | 1006.02 |
| | 8.95 | 122.10 | 1556.07 | 9 | 123.98 | 1165.54 |
| | 9.2 | 125.20 | 1870.87 | 9.15 | 125.48 | 981.38 |
| | 9.1 | 125.08 | 1856.95 | 9.15 | 125.23 | 1215.63 |
| | 8.95 | 122.62 | 1658.89 | 9.1 | 124.2 | 1016.84 |
| | 8.9 | 121.61 | 2029.61 | 8.85 | 123.19 | 944.15 |
| P6 | 8.5 | 166.37 | 2393.53 | 8.5 | 168.936 | 1331.91 |
| | 8.5 | 167.27 | 2362.11 | 8.5 | 169.18 | 1305.63 |
| | 8.5 | 168.37 | 2343.04 | 8.5 | 170.02 | 1264.41 |
| | 8.5 | 166.58 | 1631.94 | 8.5 | 168.90 | 1246.40 |
| | 8.5 | 165.94 | 2204.33 | 8.5 | 168.58 | 1421.76 |
| | 8 | 162.47 | 2364.93 | 5.5 | 167.17 | 1318.00 |
| | 8 | 161.59 | 2059.96 | 8.5 | 165.42 | 1260.41 |
| | 8 | 159.95 | 1712.13 | 8.5 | 168.44 | 1356.92 |
| | 8 | 164.3 | 2255.78 | 8.5 | 166.99 | 1370.88 |

## 8. Conclusion

Finding a satisfying configuration for the RMiMAL, where several models are assembled by a set of robots in an intermixed sequence, can decrease wasted time and save energy by eliminating the reconfiguration of the line at each entry of a new model. In this paper, the energy-efficient RMiMALBP is addressed for the first time, and to solve it, a memory based Cuckoo Search Algorithm (MBCSA) is

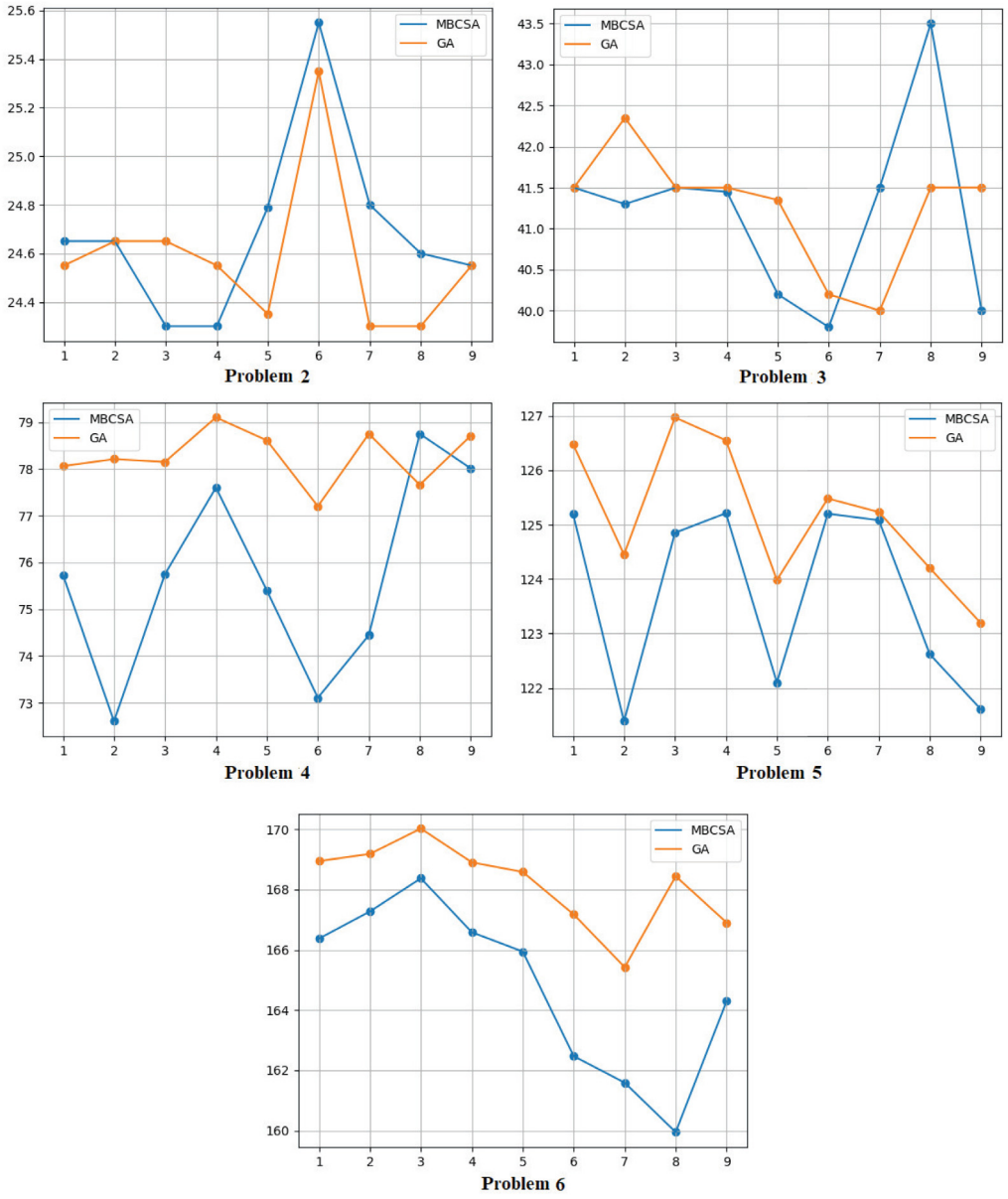



**Fig. 8.** Obtained solutions for all problems by MBCSA and GA .

proposed. Searching for good solutions to this complicated problem in a reasonable time is a big challenge, especially for large-sized problems, and for this reason, the memory technique is integrated into the MBCSA in order to get a compromise between the diversification and the intensification in the search space while solving (EEMMALBP). In order to evaluate the performance of the proposed MBCSA, six problems of different sizes were generated. The obtained results by the MBCSA were compared with those obtained by the famous Genetic Algorithm and the memory less CSA. The MBCSA and the GA obtained almost similar results for s 1,2 and 3, but for the large-scale problems 4, 5, and 6, the MBCSA outperforms the GA in terms of obtained objective values. The CPU times taken by the MBCSA while solving problems 1, 2 and 3 were higher than those taken by the GA. Also, the MBCSA outperforms the MLCSA in solving large-scale problems. The proposed MBCSA is a little slow and needs more CPU time when solving large-scale problems. This weakness is the result of the usage of memory, which helps the algorithm escape from the local optimum and continue searching for new solutions that were not discovered in previous iterations.

In future work, we will improve the MBCSA so it can solve large-sized problems with fewer CPU times by using, for example, the principles of parallelism and distribution. Furthermore, the MBCSA can be modified to minimize the energy consumed in other types of lines, such as the robotic u-shaped lines and the robotic two-sided lines.

## CRediT authorship contribution statement

**Lakhdar Belkharroubi:** Conceptualization, Writing – original draft, Software, Methodology, Data generation. **Khadidja Yahyaoui:** Supervision, Conceptualization, Revision, Validation, Methodology.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

**Table 11**
Details of the best obtained solutions by both MBCSA and GA.

| P | W | MBCSA | | | | | | GA | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | R | PT | ID | E(PM) | E(SM) | TE | R | PT | ID | E(PM) | E(SM) | TE |
| P1 | 1 | 1 | 5.5 | 0 | 5.5 | 0 | 5.5 | 1 | 5.5 | 0 | 5.5 | 0 | 5.5 |
| | 2 | 1 | 5 | 0.5 | 5 | 0.2 | 5.2 | 1 | 5 | 0.5 | 5 | 0.2 | 5.2 |
| | 3 | 2 | 4.5 | 1 | 5.4 | 0.3 | 5.7 | 2 | 4.5 | 1 | 5.4 | 0.3 | 5.7 |
| P2 | 1 | 1 | 7.5 | 0 | 6 | 0 | 6 | 1 | 7.5 | 0 | 6 | 0 | 6 |
| | 2 | 1 | 7 | 0.5 | 5.6 | 0.25 | 5.85 | 1 | 7 | 0.5 | 5.6 | 0.25 | 5.85 |
| | 3 | 2 | 6 | 1.5 | 6 | 0.45 | 6.45 | 2 | 6 | 1.5 | 6 | 0.45 | 6.45 |
| | 4 | 1 | 7.5 | 0 | 6 | 0 | 6 | 1 | 7.5 | 0 | 6 | 0 | 6 |
| P3 | 1 | 1 | 6.5 | 1 | 7.8 | 0.5 | 8.3 | 1 | 7.5 | 0.5 | 9 | 0.25 | 9.25 |
| | 2 | 3 | 7.5 | 0 | 7.5 | 0.0 | 7.5 | 3 | 8 | 0 | 8 | 0 | 8 |
| | 3 | 3 | 7 | 0.5 | 7 | 0.15 | 7.15 | 3 | 7 | 1 | 7 | 0.3 | 7.3 |
| | 4 | 2 | 4.5 | 2.5 | 6.75 | 1.80 | 8.55 | 2 | 4.5 | 3.5 | 6.75 | 2.1 | 8.85 |
| | 5 | 1 | 6.5 | 1 | 7.80 | 0.5 | 8.3 | 3 | 6 | 2 | 6 | 0.6 | 6.6 |
| P4 | 1 | 2 | 7 | 0 | 10.5 | 0 | 10.5 | 2 | 7 | 0.5 | 10.5 | 0.3 | 10.8 |
| | 2 | 1 | 7 | 0 | 10.5 | 0 | 10.5 | 1 | 5.5 | 2 | 8.25 | 1.4 | 9.65 |
| | 3 | 2 | 6.5 | 0.5 | 9.75 | 0.3 | 10.05 | 3 | 7 | 0.5 | 11.2 | 0.3 | 11.5 |
| | 4 | 3 | 7 | 0 | 11.2 | 0 | 11.2 | 2 | 7 | 0.5 | 10.5 | 0.3 | 10.8 |
| | 5 | 2 | 6.5 | 0.5 | 9.75 | 0.3 | 10.05 | 2 | 7.5 | 0 | 11.25 | 0.0 | 11.25 |
| | 6 | 1 | 6.7 | 0.3 | 10.05 | 0.21 | 10.26 | 1 | 7.2 | 0.3 | 10.8 | 0.21 | 11.01 |
| | 7 | 2 | 6.5 | 0.5 | 9.75 | 0.3 | 10.05 | 2 | 6.5 | 1 | 9.75 | 0.6 | 10.35 |
| P5 | 1 | 3 | 8.35 | 0.5 | 11.69 | 0.5 | 12.19 | 3 | 8.2 | 0.65 | 11.48 | 0.65 | 12.13 |
| | 2 | 3 | 8.1 | 0.75 | 11.34 | 0.75 | 12.09 | 1 | 7.2 | 1.65 | 10.8 | 1.65 | 12.45 |
| | 3 | 3 | 8.7 | 0.15 | 12.18 | 0.15 | 12.33 | 3 | 8.7 | 0.15 | 12.18 | 0.15 | 12.33 |
| | 4 | 3 | 8.85 | 0 | 12.39 | 0 | 12.39 | 3 | 8.85 | 0 | 12.39 | 0 | 12.39 |
| | 5 | 3 | 8.8 | 0.05 | 12.32 | 0.05 | 12.37 | 3 | 8.5 | 0.35 | 11.9 | 0.35 | 12.25 |
| | 6 | 3 | 8.35 | 0.5 | 11.69 | 0.5 | 12.19 | 3 | 8.8 | 0.05 | 12.32 | 0.05 | 12.37 |
| | 7 | 3 | 8.8 | 0.05 | 12.31 | 0.05 | 12.36 | 3 | 8.85 | 0 | 12.39 | 0.0 | 12.39 |
| | 8 | 3 | 8.65 | 0.20 | 12.11 | 0.2 | 12.31 | 3 | 8.5 | 0.35 | 11.90 | 0.35 | 12.25 |
| | 9 | 1 | 8.4 | 0.45 | 11.85 | 0.45 | 12.3 | 2 | 8.75 | 0.15 | 12.75 | 0.11 | 12.86 |
| | 10 | 3 | 8.85 | 0 | 10.85 | 0.0 | 10.85 | 2 | 8.85 | 0 | 11.76 | 0.0 | 11.76 |
| P6 | 1 | 3 | 8 | 0 | 14.4 | 0 | 14.4 | 3 | 8 | 0.5 | 14.4 | 0.5 | 15 |
| | 2 | 2 | 7.5 | 0.5 | 12.75 | 0.55 | 13.3 | 2 | 8.5 | 0 | 14.45 | 0 | 14.45 |
| | 3 | 2 | 7.5 | 0.5 | 12.75 | 0.55 | 13.3 | 2 | 8.5 | 0 | 14.45 | 0 | 14.45 |
| | 4 | 2 | 7.5 | 0.5 | 12.75 | 0.55 | 13.3 | 2 | 8 | 0.5 | 13.60 | 0.5 | 14.1 |
| | 5 | 2 | 8 | 0 | 13.60 | 0 | 13.60 | 3 | 7.5 | 1 | 13.60 | 1 | 14.6 |
| | 6 | 2 | 8 | 0 | 13.60 | 0 | 13.60 | 2 | 8.5 | 0 | 14.45 | 0 | 14.45 |
| | 7 | 3 | 8 | 0 | 13.50 | 0 | 13.50 | 2 | 7.2 | 1.3 | 11.90 | 1.43 | 13.33 |
| | 8 | 2 | 7 | 1 | 11.90 | 1.1 | 13 | 2 | 7.5 | 1 | 12.75 | 1.0 | 13.75 |
| | 9 | 2 | 7.5 | 0.5 | 12.75 | 0.55 | 13.3 | 1 | 6.75 | 1.75 | 10.5 | 1.75 | 12.25 |
| | 10 | 1 | 6.75 | 1.25 | 10.5 | 1.25 | 11.75 | 2 | 8.25 | 0.25 | 14.02 | 0.275 | 14.295 |
| | 11 | 2 | 7.75 | 0.25 | 13.17 | 0.275 | 13.44 | 2 | 8.25 | 0.25 | 14.02 | 0.275 | 14.295 |
| | 12 | 2 | 7.75 | 0.25 | 13.17 | 0.275 | 13.44 | 3 | 2.67 | 5.38 | 4.806 | 5.83 | 10.636 |

**Table 12**
Comparison of objective values obtained by the MBCSA and the MLCSA.

| Problems | MBCSA | MLCSA |
|---|---|---|
| P1 | 16.4, 16.4, 16.4, 16.4, 16.4, 16.4, 16.4, 16.4, 16.4 | 16.4, 16.4, 16.4, 16.4, 16.4, 16.4, 16.4, 16.4, 16.4 |
| P2 | 24.65, 24.65, **24.3**, **24.3**, 24.8, 24.55, 24.8, 24.6, 24.55 | **24.3**, 25.2, **24.3**, 24.55, **24.3**, 24.55, **24.3**, 25.45, 24.35 |
| P3 | 41.5, 41.3, 41.5, 41.45, 40.2, **39.8**, 41.5, 43.5, 40.0 | **41.35**, 41.5, 42.35, 41.5, 42.35, 42.5, 44.0, 41.5, 42.0 |
| P4 | 75.73, **72.61**, 75.75, 77.6, 75.4, 73.11, 78.56, 74.45, 78.75 | 78.45, 78.11, 77.01, **72.76**, 78.25, 77.2, 74.75, 76.61, 78.71 |
| P5 | 125.205, **121.39**, 124.85, 125.205, 122.105, 125.205, 125.085, 126.525, 121.605 | 123.86, 123.105, 125.145, 126.295, 123.04, 125.41, 124.55, 126.701, **123.01**, |
| P6 | 166.366, 167.266, 168.366, 166.58, 165.936, 162.466, 161.56, **159.95**, 164.3 | **162.83**, **162.83**, 163.49, **162.83**, 163.49, 163.43, **162.83**, **162.83**, 163.49 |

## References


Abdullah Make, M.R., Rashid, A.M.F.F., Razali, M.M., 2017. A review of two-sided assembly line balancing problem. Int. J. Adv. Manuf. Technol. 89 (5–8), 1743–1763. http://dx.doi.org/10.1007/s00170-016-9158-3, URL: http://link.springer.com/10.1007/s00170-016-9158-3.

Aghajani, M., Ghodsi, R., Javadi, B., 2014. Balancing of robotic mixed-model two-sided assembly line with robot setup times. Int. J. Adv. Manuf. Technol. 74 (5–8), 1005–1016. http://dx.doi.org/10.1007/s00170-014-5945-x, URL: http://link.springer.com/10.1007/s00170-014-5945-x.

Agrawal, A.P., Kaur, A., 2017. An empirical evaluation of memory less and memory using meta-heuristics for solving travelling salesman problem. Int. J. Comput. Syst. Eng. 3 (4), 228. http://dx.doi.org/10.1504/IJCSYSE.2017.089208, URL: http://www.inderscience.com/link.php?id=89208.

Álvarez-Miranda, E., Pereira, J., 2019. On the complexity of assembly line balancing

problems. Comput. Oper. Res. 108, 182–186. http://dx.doi.org/10.1016/j.cor.2019.04.005, URL: https://linkinghub.elsevier.com/retrieve/pii/S0305054819300826.

Bakar, N.A., Ramli, M.F., Sin, T.C., Masran, H., 2019. A review on Robotic assembly line balancing and metaheuristic in manufacturing industry. Kedah, Malaysia, 040005. http://dx.doi.org/10.1063/1.5121084, URL: http://aip.scitation.org/doi/abs/10.1063/1.5121084.

Becker, C., Scholl, A., 2006. A survey on problems and methods in generalized assembly line balancing. European J. Oper. Res. 168 (3), 694–715. http://dx.doi.org/10.1016/j.ejor.2004.07.023, URL: https://linkinghub.elsevier.com/retrieve/pii/S0377221704004801.

Bhattacharjee, T., Sahu, S., 1990. Complexity of single model assembly line balancing problems. Eng. Costs Prod. Econ. 18 (3), 203–214. http://dx.doi.org/10.1016/0167-188X(90)90122-X, URL: https://linkinghub.elsevier.com/retrieve/pii/0167188X9090122X.

Brandão, J., 2020. A memory-based iterated local search algorithm for the multi-depot open vehicle routing problem. European J. Oper. Res. 284 (2), 559–571. http://dx.doi.org/10.1016/j.ejor.2020.01.008, URL: https://linkinghub.elsevier.com/retrieve/pii/S0377221720300278.

Çil, Z.A., Mete, S., Ağpak, K., 2017a. Analysis of the type II robotic mixed-model assembly line balancing problem. Eng. Optim. 49 (6), 990–1009. http://dx.doi.org/10.1080/0305215X.2016.1230208, URL: https://www.tandfonline.com/doi/full/10.1080/0305215X.2016.1230208.

Çil, Z.A., Mete, S., Özceylan, E., Ağpak, K., 2017b. A beam search approach for solving type II robotic parallel assembly line balancing problem. Appl. Soft Comput. 61, 129–138. http://dx.doi.org/10.1016/j.asoc.2017.07.062, URL: https://linkinghub.elsevier.com/retrieve/pii/S1568494617304829.

Chutima, P., 2022. A comprehensive review of robotic assembly line balancing problem. J. Intell. Manuf. 33 (1), 1–34. http://dx.doi.org/10.1007/s10845-020-01641-7, URL: https://link.springer.com/10.1007/s10845-020-01641-7.

Du, K.-L., Swamy, M.N.S., 2016. Search and Optimization by Metaheuristics. Springer International Publishing, Cham, http://dx.doi.org/10.1007/978-3-319-41192-7, URL: http://link.springer.com/10.1007/978-3-319-41192-7.

Gupta, S., Deep, K., 2020. A memory-based grey wolf optimizer for global optimization tasks. Appl. Soft Comput. 93, 106367. http://dx.doi.org/10.1016/j.asoc.2020.106367, URL: https://linkinghub.elsevier.com/retrieve/pii/S1568494620303070.

Janardhanan, M.N., Nielsen, P., Li, Z., Ponnambalam, S.G., 2018. Minimizing energy consumption in a straight robotic assembly line using differential evolution algorithm. In: Omatu, S., Rodríguez, S., Villarrubia, G., Faria, P., Sitek, P., Prieto, J. (Eds.), Distributed Computing and Artificial Intelligence, 14th International Conference, Vol. 620. Advances in Intelligent Systems and Computing, Springer International Publishing, pp. 45–52, Series Title.

Kapgate, D., 2022. Efficient quadcopter flight control using hybrid SSVEP + p300 visual brain computer interface. Int. J. Human Comput. Interact. 38 (1), 42–52. http://dx.doi.org/10.1080/10447318.2021.1921482, URL: https://www.tandfonline.com/doi/full/10.1080/10447318.2021.1921482.

Li, Z., Dey, N., Ashour, A.S., Tang, Q., 2018a. Discrete cuckoo search algorithms for two-sided robotic assembly line balancing problem. Neural Comput. Appl. 30 (9), 2685–2696. http://dx.doi.org/10.1007/s00521-017-2855-5, URL: http://link.springer.com/10.1007/s00521-017-2855-5.

Li, Z., Janardhanan, M.N., Nielsen, P., Tang, Q., 2018b. Mathematical models and simulated annealing algorithms for the robotic assembly line balancing problem. Assembly Autom. 38 (4), 420–436. http://dx.doi.org/10.1108/AA-09-2017-115, URL: https://www.emerald.com/insight/content/doi/10.1108/AA-09-2017-115/full/html.

Li, X., Leung, S.C., Tian, P., 2012. A multistart adaptive memory-based tabu search algorithm for the heterogeneous fixed fleet open vehicle routing problem. Expert Syst. Appl. 39 (1), 365–374. http://dx.doi.org/10.1016/j.eswa.2011.07.025, URL: https://linkinghub.elsevier.com/retrieve/pii/S0957417411009870.

Li, Z., Tang, Q., Zhang, L., 2016. Minimizing energy consumption and cycle time in two-sided robotic assembly line systems using restarted simulated annealing algorithm. J. Cleaner Prod. 135, 508–522. http://dx.doi.org/10.1016/j.jclepro.2016.06.131, URL: https://linkinghub.elsevier.com/retrieve/pii/S0959652616308083.

Lü, Z., Hao, J.-K., 2012. Adaptive memory-based local search for MAX-SAT. Appl. Soft Comput. 12 (8), 2063–2071. http://dx.doi.org/10.1016/j.asoc.2012.01.013, URL: https://linkinghub.elsevier.com/retrieve/pii/S1568494612000397.

Mukund Nilakantan, J., Huang, G.Q., Ponnambalam, S., 2015a. An investigation on minimizing cycle time and total energy consumption in robotic assembly line systems. J. Cleaner Prod. 90, 311–325. http://dx.doi.org/10.1016/j.jclepro.2014.11.041, URL: https://linkinghub.elsevier.com/retrieve/pii/S0959652614012219.

Mukund Nilakantan, J., Ponnambalam, S.G., Jawahar, N., Kanagaraj, G., 2015b. Bio-inspired search algorithms to solve robotic assembly line balancing problems. Neural Comput. Appl. 26 (6), 1379–1393. http://dx.doi.org/10.1007/s00521-014-1811-x, URL: http://link.springer.com/10.1007/s00521-014-1811-x.

Nilakantan, J.M., Li, Z., Tang, Q., Nielsen, P., 2017a. Multi-objective co-operative co-evolutionary algorithm for minimizing carbon footprint and maximizing line efficiency in robotic assembly line systems. J. Cleaner Prod. 156, 124–136. http://dx.doi.org/10.1016/j.jclepro.2017.04.032, URL: https://linkinghub.elsevier.com/retrieve/pii/S0959652617307394.

Nilakantan, J.M., Nielsen, I., Ponnambalam, S.G., Venkataramanaiah, S., 2017b. Differential evolution algorithm for solving RALB problem using cost- and time-based models. Int. J. Adv. Manuf. Technol. 89 (1–4), 311–332. http://dx.doi.org/10.1007/s00170-016-9086-2, URL: http://link.springer.com/10.1007/s00170-016-9086-2.

Nilakantan, J.M., Ponnambalam, S.G., Huang, G.Q., 2015. Minimizing energy consumption in a U-shaped robotic assembly line. In: 2015 International Conference on Advanced Mechatronic Systems (ICAMechS). IEEE, Beijing, China, pp. 119–124. http://dx.doi.org/10.1109/ICAMechS.2015.7287140, URL: http://ieeexplore.ieee.org/document/7287140/.

Rabbani, M., Mousavi, Z., Farrokhi-Asl, H., 2016. Multi-objective metaheuristics for solving a type II robotic mixed-model assembly line balancing problem. J. Ind. Product. Eng. 33 (7), 472–484. http://dx.doi.org/10.1080/21681015.2015.1126656, URL: https://www.tandfonline.com/doi/full/10.1080/21681015.2015.1126656.

Roy, A.M., 2022. An efficient multi-scale CNN model with intrinsic feature integration for motor imagery EEG subject classification in brain-machine interfaces. Biomed. Signal Process. Control 74, 103496. http://dx.doi.org/10.1016/j.bspc.2022.103496, URL: https://linkinghub.elsevier.com/retrieve/pii/S1746809422000180.

Rubinovitz, J., Bukchin, J., 1991. Design and Balancing of Robotic Assembly Lines. Society of Manufacturing Engineers.

Sahmoud, S., Topcuoglu, H.R., 2016. In: Squillero, G., Burelli, P. (Eds.), A Memory-Based NSGA-II Algorithm for Dynamic Multi-objective Optimization Problems, Vol. 9598. pp. 296–310.

Saif, U., Guan, Z., Wang, B., Mirza, J., Huang, S., 2014. A survey on assembly lines and its types. Front. Mech. Eng. 9 (2), 95–105. http://dx.doi.org/10.1007/s11465-014-0302-1, URL: http://link.springer.com/10.1007/s11465-014-0302-1.

Sivasankaran, P., Shahabudeen, P., 2014. Literature review of assembly line balancing problems. Int. J. Adv. Manuf. Technol. 73 (9–12), 1665–1694. http://dx.doi.org/10.1007/s00170-014-5944-y, URL: http://link.springer.com/10.1007/s00170-014-5944-y.

Sree Ranjini, K.S., Murugan, S., 2017. Memory based hybrid dragonfly algorithm for numerical optimization problems. Expert Syst. Appl. 83, 63–78. http://dx.doi.org/10.1016/j.eswa.2017.04.033, URL: https://linkinghub.elsevier.com/retrieve/pii/S0957417417302762.

Sun, B.-q., Wang, L., Peng, Z.-p., 2020. Bound-guided hybrid estimation of distribution algorithm for energy-efficient robotic assembly line balancing. Comput. Ind. Eng. 146, 106604. http://dx.doi.org/10.1016/j.cie.2020.106604, URL: https://linkinghub.elsevier.com/retrieve/pii/S0360835220303387.

Xin-She, Y., 2014. Nature-Inspired Optimization Algorithms. Elsevier, http://dx.doi.org/10.1016/C2013-0-01368-0, URL: https://linkinghub.elsevier.com/retrieve/pii/C20130013680.

Xu, W., Xiao, T., 2008. Mixed model assembly line balancing problem with fuzzy operation times and drifting operations. In: 2008 Winter Simulation Conference. IEEE, Miami, FL, USA, pp. 1752–1760. http://dx.doi.org/10.1109/WSC.2008.4736263, URL: https://ieeexplore.ieee.org/document/4736263/.

Yadav, A., Agrawal, S., 2022. Mathematical model for robotic two-sided assembly line balancing problem with zoning constraints. Int. J. Syst. Assur. Eng. Manag. 13 (1), 395–408. http://dx.doi.org/10.1007/s13198-021-01284-8, URL: https://link.springer.com/10.1007/s13198-021-01284-8.

Yang, X.-S., Deb, S., 2009. Cuckoo search via Lévy flights. In: 2009 World Congress on Nature & Biologically Inspired Computing (NaBIC). IEEE, Coimbatore, India, pp. 210–214. http://dx.doi.org/10.1109/NABIC.2009.5393690, URL: http://ieeexplore.ieee.org/document/5393690/.

Younes, Z., Alhamrouni, I., Mekhilef, S., Reyasudin, M., 2021. A memory-based gravitational search algorithm for solving economic dispatch problem in micro-grid. Ain Shams Eng. J. 12 (2), 1985–1994. http://dx.doi.org/10.1016/j.asej.2020.10.021, URL: https://linkinghub.elsevier.com/retrieve/pii/S2090447921000277.

Zhang, Z., Tang, Q., Li, Z., Zhang, L., 2019. Modelling and optimisation of energy-efficient U-shaped robotic assembly line balancing problems. Int. J. Prod. Res. 57 (17), 5520–5537. http://dx.doi.org/10.1080/00207543.2018.1530479, URL: https://www.tandfonline.com/doi/full/10.1080/00207543.2018.1530479.