Efficient Application of Metaheuristic Algorithms for Balancing Human Robot Collaborative Assembly Lines with Case-study validations

Ponnambalam S, Sridharan A P and Ranganathan S V

ABSTRACT

With the advent of industry 5.0 practices, resilient production technologies are advancing progressively due to increasing emphasis in shifting towards sustainability and efficiency. Human-robot collaboration (HRC) production methods are specifically eyed upon for this purpose, due to the ideology of leveraging the flexibility, intellect, consistency and accuracy of robot and human combinations. Given the sophisticated possibilities of resource allocation with multiple optimization objectives when solving semi-automated robotic assembly line balancing (RALB) problems in large-scale industries, this paper presents a realistic study in solving energy-oriented line balancing problems with collaborative robots in a StAL (straight line assembly line), for a single product type. In addition to the previously considered studies in the area of HRC, this study addresses several task and workstation constraints with a sense of balanced consideration of robustness, ergonomics, safety and an evenly spread semi-automated environment, to reduce cycle time and line energy consumption. Four problem-suited algorithms identified, namely, Teaching-learning-based optimization-TLBO, Particle swarm optimization-PSO, Migrating birds optimization (MBO) and a recent Archimedes optimization algorithm (AOA) are implemented using modified discretization procedures for updation of search. The algorithms are hybridized with the scout phase from the Artificial bee colony-ABC algorithm, blending its associated advantages for yielding better pareto fronts by periodically replacing any recurring solutions and thus increasing exploration and exploitation spaces. Moreover, new fine tuning methods are proposed for all the developed hybrid algorithms in order to minimize computational efforts. The Pareto optimal solutions of the four hybrids and their standard algorithms are collectively compared and illustrated using various performance indicator plots, on 32 standard testing HRC datasets of varying sizes and workstations. The study indicated that the hybridized AOA algorithm, which was never used for solving ALBP, performed slightly better than others and needed significantly less computation. A case study is carried out to validate the working and results of the developed line balancing methodology in an electronic chip manufacturing industry.

KEYWORDS

Assembly line balancing; Human–robot collaboration; Collaborative robots; Multi-objective optimization; hybridization; metaheuristics

1. Introduction

The manufacturing industry is constantly driven by challenges in adapting to the rapid rise of new and innovative technologies. As the demand curve increases tremendously due to updation and induction of new utilities in every product line, the industries that manufacture, assemble and process such products find a dire need to be quick and efficient to keep up with the market. Among the various processes involved in a product supply chain, the assembly process involved is regarded as a separate module to be assessed when dealing with process optimization. The assembly process is the final step in the whole production process, after which a finished product is witnessed. This chapter is presented in an orderly fashion, with the current research areas in balancing assembly lines. It takes us through the common ideology of assembly line balancing, the current trends of automation involved in assembly lines. As this study is based on HRC or manual & automation resources utilization, an outline of the current human-robot collaboration practices are then given. Some of the popular metaheuristic algorithms that are being used currently in the HRC line balancing research are investigated for their variant improvements and suitability. Lastly, the gaps identified, the objectives and assumptions made in this study are discussed.

1.1 Assembly Line Balancing

Assembly line balancing (ALB) is a method of optimization in which the overall assembly workload is divided in such a manner that each task in the workload follows precedence relationships and their allocation to workstations and the available set of resources yield the least amount of lead time to complete the entire assembly process. One might consider it analogous to the traveling salesman problem for optimizing transportation problems. In any industry, the optimization of assembly lines can also start with identifying and balancing several sub-assembly processes in order to break down the study complexity or in automating a set of tasks under economic considerations. Hence, the goal of balancing assembly lines varies according to the problem environment. In

considering the current approaches to solving ALB problems, the first category deals with maximization of throughput in the assembly line with a given amount of resources **(Mura, 2019)**. Whereas, the second category is contrary to the first and deals with minimization of cost of resources for a fixed throughput number (**Boysen, 2021**). Moreover, different layouts of assembly lines exist in real life such as two-sided assembly lines, circular lines, multi manned, parallel workstation types, U-shaped lines, straight line (most common), parallel U-line type, parallel adjacent U-line, parallel multi-manned, and so on, out of which many often deal with mixed-model assembly. For instance, the studies conducted by (**Hamzadayi, 2018**) **and** (**Qiuhua, 2017**) demonstrate the necessity of balancing two-sided assembly lines, where large assembly models require work-handling in every direction and the orientation setup times are very large, which may prove infeasible. This shows that the best layout is mostly dependent on the type and count of products being assembled.

1.2 Robotic Assembly Line Balancing

In recent years, assembly processes in industries have eloped from traditional assembly lines with manual labor task implementations to robotic process automation systems. With the introduction of AI and commercial automation technology, the simple justification to replace them with their manual counterparts is due to their accuracy, speed, flexibility and long-term cost benefits involved **(Malik, 2019)**. The robotic assembly lines, therefore lead to high volume and stable production levels. From the knowledge gained during the industrial case study implementation done later in this paper, the initial cost of automation deployment is on terms such that the cost is recovered within 20% of the total project timeline. Hence, a vital part of robotic ALB also lies in automating existing manual processes by considering ROI and task feasibility. The general ALB problem has various objectives such as minimizing cycle time, cost, increasing safety, ergonomics etc.., with constraints such as task precedence relations, number of workstations, resources etc. As industrial robots and gantries are accurate in performing tasks, they are assumed to have deterministic operation times in solving RALB problems (**Chutima, 2022**). Some other assumptions made by the researchers on solving them were: (1) The precedence relation of tasks are known and each type of robots have their own associated operation times. (2) One workstation comprises only one robot and all robot models are readily available to use without any capacity and cost limitations. Furthermore, several approaches have been categorized as to balancing robotic lines. The RALB type-I aims to reduce the workstation count by assigning best suited robots and tasks to workstations; RALB type-II aims at minimizing the cycle time with predefined workstation counts; RALB type-E aims at reducing cycle time and workstation count together, thus maximizing the assembly line efficiency; RALB type-F aims at finding feasible solutions for a predefined set of workstations and cycle times; RALB type-COST aims at monetary and economic aspects; RALB type-O involves other categories not listed above **(Tamás, 2021), (Weckenborg, 2020)**.

1.3 Human-Robot Collaborative Assembly Line Balancing

To combine the flexible decision making of manual labor with the accuracy of robots, the study of line balancing in several workspaces considers both humans and robots working hand in hand. With increase in complex part integrations and compactness of systems, modern workspaces require a fail-proof environment to maintain efficiency and repeatability. Thus, a collaborative line balancing results in a semi-automated assembly line where human, robot and human-robot (COBOT) resources are allocated among workstations and assigned with appropriate tasks. This method of allocation can induce a great extent of intelligent manufacturing capability and flexibility to assembly systems **(Weckenborg, 2020), (Ana, 2021)** . With the well known fact that employing robots reduces lead time and that employing manual labor accounts for assurance and safety, this paper considers the several constraints and measures pertaining to each individual task execution by collaborative resources. Berx et al. **(Nicole, 2021)** suggests careful assessment of every individual task as a primary step before deciding on its implementation ability by either humans, robots or cobots. By also keeping in mind the cost of automation and margin of lead time reduction, the ranking of risks associated with every task and resource combinations can be documented, to further decide on alternative allocation criteria and help strike a perfect balance. The known possibilities of human-robot interactions in a workstation can be stated by considering the following scenarios: (1) If the human worker is present in the opposite lane to the robot, where both work on the same task; (2) If the human and robot are in a single workstation, but work in shifts and don't coexist; (3) If both human and robot are present in a single workstation but do not work simultaneously; (4) If both the resources work simultaneously/does parallel work **(Gualtieri, 2021)**. Setting aside the first two scenarios, scenario (4) is largely neglected by several prominent studies in this field **(Amir, 2022), (Li, 2021)**. This is due to safety considerations that parallel tasking of both humans and robots may overlap and cause disruption. However, it is worthwhile to point out that the extent of error possibility largely depends on the sensitivity of tasks in the workspace. Additionally, parallel working lessens the interaction between automation and humans, thus increasing the error occurrence possibility due to reduced manual moniterance **(Mura, 2022)**.

1.4 Multi-objective Metaheuristic Algorithms

Metaheuristics provide excellent methodologies in solving optimization problems that involve a multitude of datasets. It remains an active area of research, where new updation mechanisms are being proposed based on natural observations. The optimization issues that have aroused the interest of such techniques range widely in complexity. It varies from continuous to discrete

computations and provides the flexibility of including constraints within its mechanism. Because of their intricate nature, solving these challenges is not an easy process. Exact algorithms are usually non-polynomial in nature and, although delivering the best results, have prohibitive execution durations and/or processing requirements for big data sets. Metaheuristic algorithms are meant to find approximate/ optimal solutions in realistic execution times for NP-hard optimization problems and give a practical and elegant answer to many such situations **(Tansel, 2019)**. In general, metaheuristics draws upon its motivation from nature-based interaction techniques such as the collective swarm behavior among animals or the daily interactions towards improvement or evolutionary techniques per say. A large portion of studies that solve ALB problems use these optimization mechanisms to allocate tasks and resources by considering large amounts of task-resource-constraint combinations as input data to such algorithms. The few new generation metaheuristic algorithms that have been considered for this study include PSO, TLBO, MBO and AOA as their embedded convergence mechanisms provide great flexibility in modifying or creating variants, in order to suit the specific problem that is being considered. For instance, the different PSO variant algorithms devised through Rao et al. **(Rao, 2021)** demonstrated an induction of a new phase/strategy that further improves the performance of the standard algorithm or simply covers a gap of possibilities between different existing metaheuristic algorithms. AOA is picked for study considerations solely due to its absence in discrete optimization applications, and this study demonstrates its performance and stance with other popularly used algorithms.

2. Literature Review

Several different scenarios of prerequisite input conditions/constraints are considered for every ALB research problem, some of which include precedence constraints, complexity ranking and classification of tasks, operational and setup times, assumptions of resource availability or other constraints, all in the interests of the type of assembly line under study **(Ana, 2021)**. Research emphasis is still downside on data retrieval models that serve as a foundation to formulating line balancing problems. Predetermined motion time systems are widely employed as precedence data retrieval methods that aggregate elementary motion types involved in any products' assembly (gripping, lifting), by considering their individual weighted average operation times as well as ergonomic work conditions. Moreover, machine learning predictive models can be useful in deciding the "at least near" precedence relation sequence, to aid as input data for algorithmic ALB (**Boysen, 2021**). With the advent of automation, robotic assembly lines were introduced and robotic line balancing studies created a new trend in the research area of RALB. The use of cobots in the assembly line showed better results and gained interest by many industries. As this area of research is quite new, only a few papers have been published since 2019. Many real life constraints such as tools assignment, tool space optimization, etc.., were not addressed in most of the papers and this could be the future in the study of RALB. Also only a handful of studies addressed the environmental issues (carbon footprint) by the robots (**Hamzadayi, 2018**). The existing RALB lines are of different layouts, such as MAL(multi manned assembly line), PWAL(parallel workstation), PAL(parallel), UAL(U-shaped), StAL(straight line), 2SAL(two sided), PUL(parallel U-line), PAUL(parallel adjacent U-line), PMAL(parallel multi-manned). The concept of 4 M's are kept in mind while considering the several resources of an assembly line, Man, Machine, Material & Method **(Leonardo, 2018)**. Several methods are used to arrive at a solution set, and some of them are exact optimization techniques, using heuristics, meta-heuristics, etc (**Chutima, 2022**). As the automation in the manufacturing department is increasing significantly, several robots are developed to carry out several tasks. However, not all tasks can be automated due to the lack of flexibility with available robotic technologies. Humans are flexible, adaptable according to market demands, and also have decision making skills with creativity.For the assignment of the cobots to different stations with balanced distribution of workload to both human workers and the robotic partners, MILP methods are widely used although it sets a major drawback in problem size limitations **(Weckenborg, 2020)**.

The major concern with automation is their purchasing costs **(Li, 2021)**, which indirectly correlates with the utilization efficiency and power consumption. Most industries ensure that the procurement costs are worth the investments on remunerative contract projects they deal with and also ensure the terms of ROI in the near term. As most portion of the literature only considers non-parallel working of both human and robot in a single workstation, the study conducted by Zhang et al. **(Zhang, 2022)** facilitates a stochastic line balancing model where a portion of the tasks are allowed parallel work handling between the two resources. This can prove to be the bare minimum requirements that can be considered in balancing studies that might want the parallel working of humans and robots on the same component. The recent studies in HRC ALB areas focus on three allocation possibilities: (1) only workers are present in workstations; (2) either workers or robots are present in the same workstation; (3) both workers and robots are present in a workstation. For multi-objective optimization approaches for the same, either the idealistic objective value is fixed and the required resources are determined or alternatively, the optimal objective value is determined for the available amount of resources (**Tamás, 2021**).

During task allocation between robots and humans, skill requirements for all tasks are assessed in order to assign robots with tasks pertaining to low skill requirements, high complexity, precision and repetitiveness; and assign human workers with ergonomic tasks of little complexity and high cognitive adaptability skills to adapt to an error-free operative environment and suppress the risk of automation **(Nicole, 2021), (Weckenborg, 2019)**. Metaheuristics largely aid in quick computations of line

balancing problems that consider several constraints and combinatorial possibilities in simultaneous task and resource allocation along the assembly lines **(Tansel, 2019)**. Population based metaheuristic algorithms prove superior performances **(Rao, 2021)** and also a long chain of multiple line balancing implementations that have been carried out over the years **(Amir, 2022),** (**Qiuhua, 2017**), **(Li, 2021), (Rashid, 2016)**. Similar to line balancing, studies have demonstrated several other optimization applications in various fields **(Zhang, 2020), (Kashani, 2021), (Parsopoulos, 2008)** which mainly used the PSO and TLBO variants. These studies introduce variants and modifications to the standard algorithms to better suit the specific problem environment or to improve the existing mechanisms **(Rao, 2021), (Ying, 2019)**. The basic intention behind introduction of modifications by combining various other nature inspired meta-heuristic approaches are to: (1) Enhance exploration; (2) Converge to optima in an appropriate time to improve efficiency; (3) Avoid premature convergence towards local optima; (4) increase the overall logical suitability of the approach, based on the optimization problems. In finding optimal solutions in multi-objective optimization problems, the NSGA-II technique is the most widely used, with applications in novel HRC studies. For this reason, this study also utilizes a reduced computational form of NDS as given in the NSGA-II formulations **(Deb, 2002)**.

By far, it is clear from literature that studies on collaborative line balancing lack considerations of line energy consumption, which is a key factor in determining manufacturing costs and efficiency. The few existing studies on HRC focus very little on the complex discretization methods that are often raveled upon while decoding the problem environment through metaheuristic algorithms.

3. Problem Description

The problem is formulated through literature and realistic considerations of line ergonomics, safety and intellectual diversity which is supported by a case-study done in the electronics industry. As like any other HRC line balancing problem, this problem takes into account three possibilities of assembly line work setting: tasks operated by a human alone, tasks operated by concerned robots alone, and tasks operated by both human worker and robot in collaboration (termed as "cobot" in our problem). As in any industry, all HRC resources (humans/individual robots and cobots) considered for this problem are strictly constrained based on their abilities/possibilities in carrying out each task. As shown in **table 1**, only the task operations assigned with '1' can be carried out by the respective resource.

After careful considerations from literature and case-study, and to not neglect any possibility and choice for readers, two scenarios in the HRC resource availability have been considered for this problem: 1) The primary consideration is such that, there is an infinite count of robots of required types and human workers available for allocation among the given number of workstations. 2) There is a known count of robots of each type and an infinite count of human workers.

Table 1. Task Constraints for HRC Resources

TASK CONSTRAINT	TASK 1	TASK 2	TASK 3	TASK 4	TASK 5
ROBO 1	1	1	1	1	0
ROBO 2	1	0	1	0	1
ROBO 3	1	1	1	0	0
COBOT 1 (robot 1 + human)	1	1	1	1	1
COBOT 2 (robot 2 + human)	1	1	1	1	1
COBOT 3 (robot 3 + human)	1	1	1	1	1
HUMAN	0	1	0	1	1

Although the first scenario lacks economical considerations to an extent, it is followed by a resource allocation constraint for the given number of workstations. This constraint is introduced to develop an optimized allocation sequence for an evenly spread semi-automated assembly line. The constraint is such that no two consecutive workstations shall have the same resource configurations. **Figure 1** shows an assembly line illustration where the work station constraint is considered. Each of the three possibilities depict a different possible combination of resource allocation, in all of which no two consecutive stations have the same type of resource.

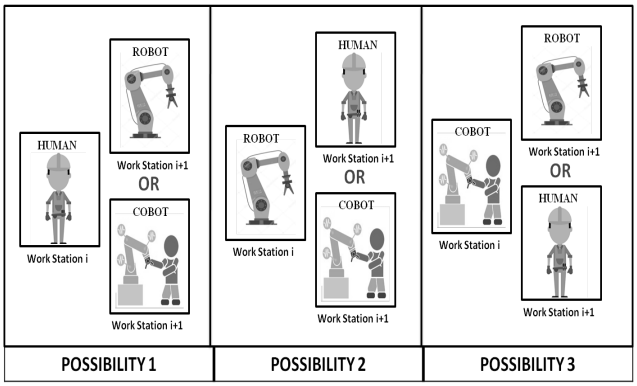

Figure 1. Illustration of proposed workstation constraint

Based on the problem considered by Li et al. **(Li, 2021)**, the process alternatives sequence shall be used to execute the resource allocation constraint successfully. Regarding several task executions within a single workstation, it is to be noted that this problem considers only one type of resource allocation per workstation (either one human, one robot or one cobot) and neglects the parallel working of cobot elements, which would otherwise compromise robustness in the light of anomalies. The problem could, however, accommodate parallel working of cobot elements under management considerations. For this problem, data pertaining to prior known operational times,

setup times and so forth are randomly calculated based on upper and lower bound values, as to account for task time fluctuations in the assembly lines and also to reduce fatigue in real-world HRC data generation.

Given below are the assumptions considered for this RALBP TYPE - II problem, considering the line characteristics.

- This study considers only a straight line assembly line (StAL) where a single model is assembled.
- Number of assembly workstations is known prior.
- Skill sets of all human workers are the same.
- Task-possibilities data and individual operation times of all types of robots and human workers in terms of upper and lower bounds are known prior.
- Sequence dependent times and setup times of all resources are known prior.
- Precedence relations of all tasks are known.
- Energy ratings of robots are known (can be considered for cost analysis along with labor charges of human workers)
- Robot maintenance is assumed to be done during non-production hours. In the event of any robot breakdown/down-time, the workstation(s) equipped with the failed robot(s) are simply removed to generate a new temporary emergency task allocation sequence with the humans.

4. Data Generation Methodology

In line with the basic data requirements in solving assembly line problems in HRC environment, some of the inputs that are considered are: 1) Total number of tasks; 2) Total workstation count; 3) Number of different robot models; 4) Task operation times for each robot models, individual human worker, and cobot; 5) Setup times and sequence dependent times for each robot model; 6) Precedence relation of all tasks; 7) Energy rating of every robot models. Due to the lack of standardized/established datasets for HRC data models, a module [C++ code] is developed for the generation of data according to the methodology below, which also conforms to the norms of the HRC problem environment. To generate the above stated data for this problem, unique methodologies are proposed as follows:

- Number of different cobot models = Number of different robot models.
- A cobot model [type i] is equivalent to a robot [type i] working with a human worker.
- Only one type of human worker is assumed (all workers are equally skilled).
- The lower bound (lb) and upper bound (ub) of operation time for each robot model is inputted by the user and is used for generating all the remaining data (operation time for cobot & human, setup and sequence dependent time of each type of robot).
- With the consideration of human worker flexibility, the setup time and sequence dependent time for human workers are neglected.

The bounds are generated using four parameters, a1, a2, a3 & a4 whose values belong in the range [0,1]. **Table 2** shows the equations devised in calculation of the extreme boundary values. For generating the range for human workers, the average lower bound and upper bound of all the types of robots is used [lb_a = average of lower bounds of all types of robots & ub_a = average of upper bounds of all types of robots].

Table 2. Proposed equations for data generation

DESCRIPTION	FORMULAE
Lower limit of task time for robot [type i]	lb_i
Upper limit of task time for robot [type i]	ub_i
Lower limit of setup time for robot [type i]	a1 * lb_i
Upper limit of setup time for robot [type i]	a1 * ub_i
Lower bound of sequence dependent time for robot [type i]	a2 * lb_i
Upper bound of sequence dependent time for robot [type i]	a2 * ub_i
Lower limit of task time for cobot [type i]	lb_i - (a3 * (ub_i - lb_i))
Upper limit of task time for cobot [type i]	ub_i - (a3 * (ub_i - lb_i))
Lower limit of task time of human	lb_a + (a4 * (ub_a - lb_a))
Upper limit of task time of human	ub_a + (a4 * (ub_a - lb_a))

As stated earlier, for the purpose of comparing the performances of the various hybridized algorithms with standardized HRC datasets developed in this paper, the manual entry of exact data pertaining to the assembly line resource's operation times, setup times and so forth are laborious. Instead, randomized data generation with reference to the upper and lower bound calculations given in **table 2** are considered. However, the manual entry of robot model/cobot specific data can be accounted for by modifying the algorithm accordingly.

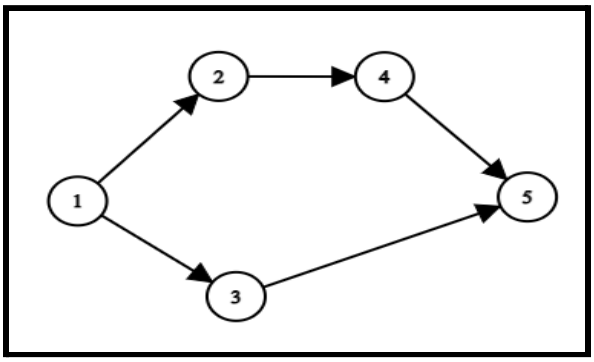

Figure 2. Precedence diagram for a 5 task problem

A set of tabulations given in this section provide an instance for data generation of a 5 task scenario, with 3 different robot models. The precedence relations of tasks are illustrated in **figure 2** and whose numerical representation is shown as in **table 3** below. The parameters a1, a2, a3 & a4 are instantiated in the beginning and are taken as 0.25, 0.1, 0.5 & 0.3 respectively, in order to perform the necessary calculations. Upon manual entry of lower and upper

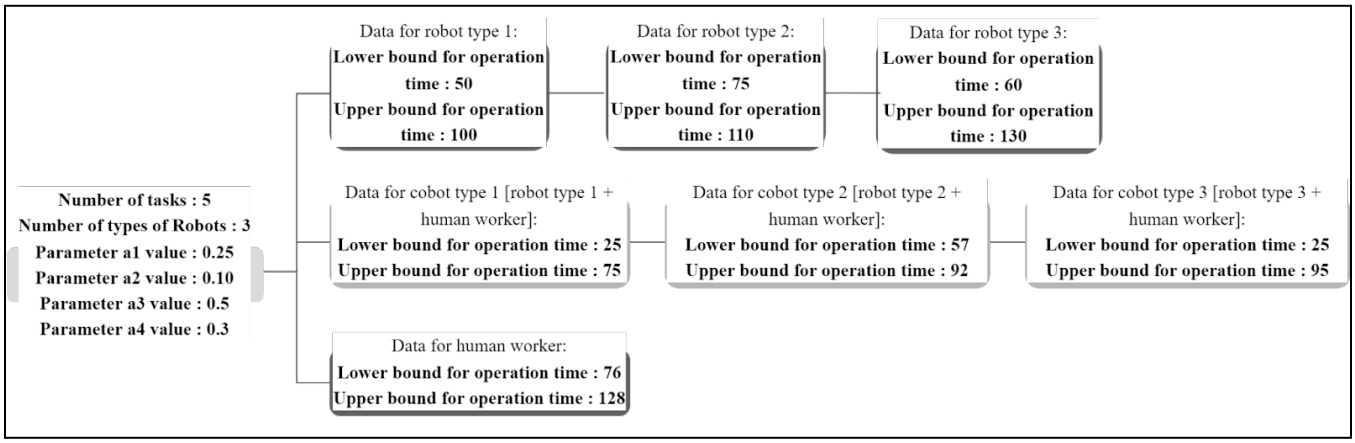

Figure 3. Illustration of example input values for data generation

bound time values for each of the 3 robot models as shown in **figure 3,** the algorithm is set to generate the operation times of each robot model, cobot pairs and the human worker, as shown in **table 4**. The null spaces in the table containing different operation times signify the impossibleness of task implementation by a particular resource for a given task. As shown in **table 5** & **table 6,** the setup times and sequence dependent setup times for every resource type are displayed with the help of the range bound equations considered. Additionally, the energy rating tabulation is presented as shown in **table 7.**

Table 3. Precedence matrix tabulated from task relationships

PRECEDENCE MATRIX	*Task 1*	*Task 2*	*Task 3*	*Task 4*	*Task 5*
Task 1	0	1	1	0	0
Task 2	0	0	0	1	0
Task 3	0	0	0	0	1
Task 4	0	0	0	0	1
Task 5	0	0	0	0	0

Table 4. Task operation times of HRC resources (example data)

OPERATION TIME	*Task 1*	*Task 2*	*Task 3*	*Task 4*	*Task 5*
Robot 1	56	68	82	94	---
Robot 2	76	---	93	---	90
Robot 3	102	111	78	---	---
Cobot 1 (robot 1 + human)	48	70	71	48	74
Cobot 2 (robot 2 + human)	67	76	85	80	60
Cobot 3 (robot 3 + human)	74	62	68	26	29
Human	---	117	---	96	89

Table 5. Setup times of HRC resources (example data)

SETUP TIME	*Task 1*	*Task 2*	*Task 3*	*Task 4*	*Task 5*
Robot 1	24	23	13	19	---
Robot 2	18	---	---	21	21
Robot 3	22	24	29	---	---

Table 6. Sequence dependent times of HRC resources (example data)

SEQUENCE DEPENDENT TIME FOR ROBOT 'i' (i=1)	*Task 1*	*Task 2*	*Task 3*	*Task 4*	*Task 5*
Task 1	7	7	8	5	8
Task 2	8	8	7	6	7
Task 3	6	10	8	6	9
Task 4	5	10	5	10	10
Task 5	5	7	10	6	5

Table 7. Energy consumption values of HRC resources (example data)

ENERGY CONSUMPTION	*Robot 1*	*Robot 2*	*Robot 3*	*Human*
Units in [kW]	0.2	0.18	0.25	0.1

5. Encoding and Decoding

Whilst there are a multitude of task sequence combinations, only a handful of them hold the optimal values of both the two objectives (cycle time & energy) considered in this study. The encoding procedure for this study is carried out based on task sequence vectors and process alternative vectors. Each task sequence vector

represents the combinations of task numbers ordered randomly and which strictly follow precedence relations. The process alternative vector contains the resource allocation identity numbers [1-human/ 2-Robot of type i/ 3-Cobot of type i] for each workstation. For this study, the process-alternative vectors follow the workstation constraint mentioned in the problem description by not having the same identity number in any two consecutive vector digits. The generation of task sequences for a given number of sequence combinatorial populations is such that the preceding task(s) of every task is stored in an adjacency list and is then used for generating the sequence using topological sorting **(Rashid, 2016)**. The topological sort procedure is as follows:

Algorithm 1 Detailed Procedure for Generating Task Sequence

1: **n = 0;** number of tasks in the generated sequence (initially no tasks are in the vector)
2: **N;** total number of tasks that are to be allocated
3: Create an empty vector - **task sequence vector;** holds the generated task sequence
4: Create an **available set;** consists of tasks that has no precedence
5: **While** (n <= N)**:**
6: Pick a random task from the available set
7: Insert this task into the task sequence vector
8: Remove this task from the available set
9: Insert the tasks that had this removed task as its precedence
10: Increment n by 1
11: **End While**

An example in **figure 4** portrays encoding of both task sequence and process alternative sequence, considering the workstation constraint mentioned earlier. To simultaneously account for the two objectives (cycle time & energy), it is necessary to effectively decode them from the above encoded vectors and calculate the respective operational times. For the calculation of cycle time after the allocation of a particular task sequence and process alternative among workstations, a consecutive method is carried out which accounts for both the mentioned encoding vectors. The procedure starts with the evaluation of initial cycle time (CTi), being the least possible cycle time that can be yielded. This is done by adding the operation times of all tasks, which are done by a particular HRC resource that takes the shortest time in carrying out the respective individual tasks. While allocating tasks to all workstations, this CTi is incremented by 1 until we get a feasible allocation of all the tasks in the workstations. Therefore, as shown in **figure 8** tasks allocated in any individual workstation are such that their combined operational times are less than CTi at any point of time.

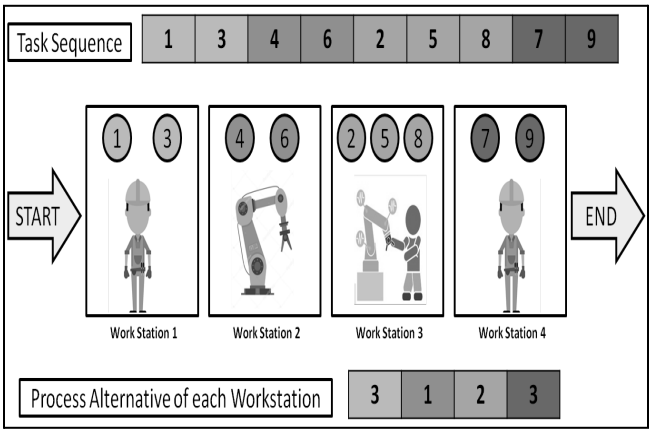

Figure 4. Encoding criteria for task and resource allocation

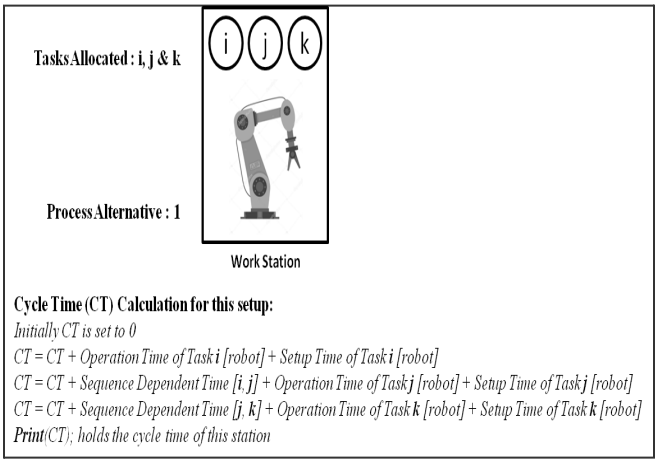

Figure 5. Cycle time calculation for robot allocation (process alternative = 1)

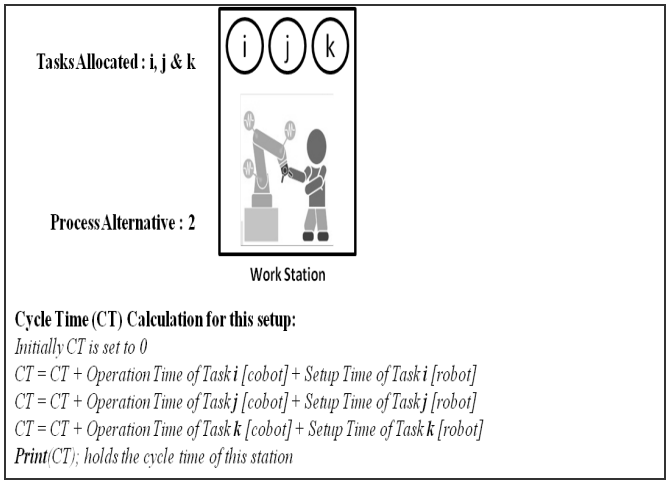

Figure 6. Cycle time calculation for cobot allocation (process alternative = 2)

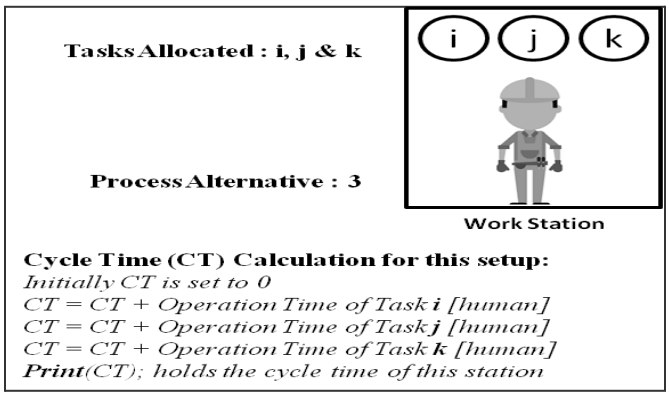

Figure 7. Cycle time calculation for human allocation (process alternative = 3)

Algorithm 2 Detailed Consecutive Method Procedure for Calculating Cycle Time

```
1:  CT; initial cycle time
2:  st = 0; start index
3:  For each workstation in order:
4:      ct_w = 0; cycle time of particular workstation
5:      pa_w; initialized from the process alternative vector
6:      While(ct_w < CT):
7:          If (pa_w is 1):
8:              For i of each type of robot:
9:                  ct_i = 0; cycle time of robot type 'i'
10:                 Check task at index 'st' in the task sequence
                    vector can be done by the robot of type_i
11:                 If (the robot can't do the task):
                            Go for the next type of robot
12:                 Else:
                            Allocate this task to the workstation
                            Increment st by 1
                            Increment ct_i according to the task
                            allocated
13:                 End if
14:             End for
15:             After trying all possible types of robots, pick
                the best robot, ct_w = minimum of all (ct_i)
16:         End if
17:         If (pa_w is 2):
18:             For i of each type of cobot:
19:                 ct_i = 0; cycle time of specific cobot type
20:                 Allocate this task to the workstation
                    Increment st by 1
                    Increment ct_i according to task allocated
21:             End for
22:             After trying all possible types of cobots, pick
                the best cobot, ct_w = minimum of all (ct_i)
23:         End if
24:         If (pa_w is 3):
25:                 Allocate this task to the workstation
                    Increment st by 1
                    Increment ct_w according to task allocated
26:         End if (% To check if all tasks are allocated %)
27:         If ( st < N [total number of tasks] ):
                    st = 0
                    CT = CT + 1
28:         End if
29:     End while
30:  End for
```

Using the pseudocode given in **algorithm 2**, cycle time for each workstation is calculated and the final CT (obtained after running the algorithm) is considered as the cycle time of the given task sequence and process alternative vector. Parallelly, the energy consumption is calculated as part of the secondary objective by multiplying the energy rating of the process alternative with the cycle time of each workstation and is summed up, as shown in **figure 8.**

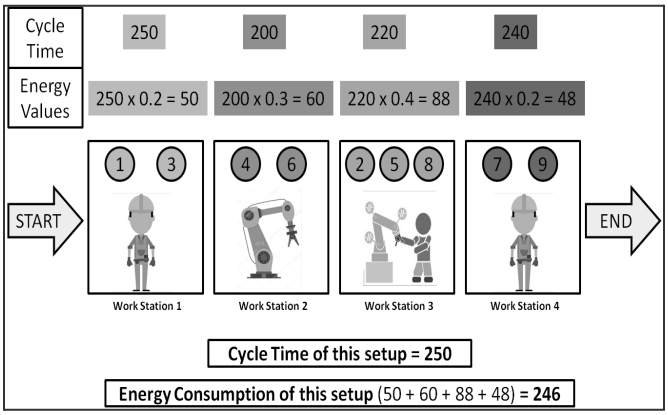

Figure 8. The cycle time & energy consumption values for the retrieved allocation from generated input data

6. Proposed multi-objective algorithms

In this study, four novel population-based algorithms are described for their working and later on, are presented with effective hybridization implementations that enhance their respective performances of exploration and exploitation in a balanced manner. The four proposed algorithms, namely PSO, TLBO, MBO and AOA, are particularly drawn upon for solving multi-objective energy-oriented semi-automated RALBP as they strike intuitively analogous to real world functioning of transitioning towards improvement/ optimization with numerous possibilities. PSO has extensively been used to solve geotechnical problems and the algorithm could be implemented by a vast number of variant approaches to suit many problem environments **(Kashani, 2021)**; TLBO has a relatively unexplored research grounds in solving multi-objective HRC line balancing problems and contains less parameters to carry out computations. The MBO algorithm is easily understood, recently implemented for discrete variables and whose performance stands on par with novel algorithms **(Li, 2021)**. AOA is the most recently introduced algorithm of all, that hasn't been used in discrete environments and assembly line balancing areas. It needs no separate fine-tuning methods and is found to perform better than most algorithms till date **(Hashim, 2021)**. Moreover, all these algorithms extend their simplicity to be discretized and hybridized with other popular algorithm fragments, thus gaining marginal advantages.

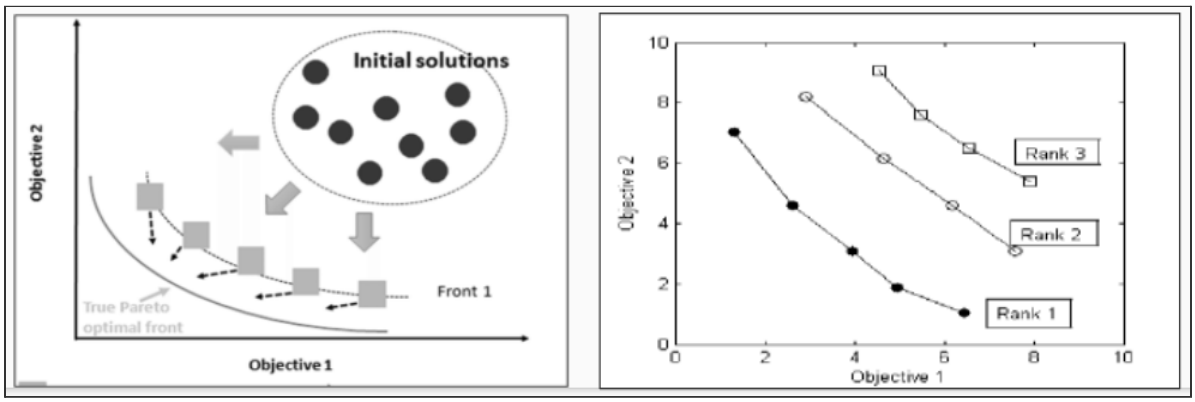

Figure 9. Position of front 1 solutions w.r.t both objectives

During the transition towards finding the best solution (population member) as the search evolves in each iteration of the algorithm, the elite and fast non-dominated sorting (NDS) approach by Deb et al. **(Deb, 2002)** is employed to embed an effective multi-objective approach in the updation and convergence of the search population. As compared to a single-objective approach where there exists only a single feasible solution, a multi-objective approach can have conflicting situations where more than one seemingly feasible solution can exist. Therefore, this study employs the use of a fast and elitist approach, where an external archive (EA) or a Pareto-optimal set holds the several feasible solutions. As shown in **figure 9,** the front 1 solutions lie on the first curve, near to the origin. These front 1 solutions obtained are referred to as the Pareto-optimal solutions, which are stored in the EA and compared to several other feasible front 1 solutions obtained from performing NDS, from each iteration. The EA is constantly updated by in turn determining the front 1 solutions from the existing front 1 solutions of each successive iteration. This can be seen in the external archive updation mechanism, under the pseudocodes of each algorithm mentioned below in **algorithm 4, Algorithm 5, Algorithm 6, Algorithm 7.**

Till date, studies to tackle problems based on multi-objective RALB using the NDS approach **(Li, 2021), (Rashid, 2016)** were done by calculation of all possible frontal solutions (front 1, front 2, ...), leading to time complexities. After empirical trials in determining the best solutions among the population after each iteration using the NDS approach, it was found that only the front 1 solutions are sufficient to be calculated and regarded as the Pareto-optimal set. Therefore, this study uses a modified approach to perform NDS by calculating the front 1 solutions alone, using a domination criteria as shown below in algorithm 3. The front 1 solution members from the newly generated population after each iteration are found by comparing the objective values of each individual population member separately with all other members of the population. Any individual population member is said to dominate the other if its cycle time and energy objectives are "better" than the compared rest. If any member dominates all other counterparts, then it is considered as a front 1 solution.

Algorithm 3 Pseudo code for NDS
(Selecting one Best Solution)

```
1:  Front1; stores the front 1 solutions
2:  For i=1 to P (population size)
3:      N_i = 0; stores the number of solutions that is
        better than i'th solution in the population data set
4:      For j=1 to P (population size)
5:          If(i is same as j):
                Skip to next j
                (to avoid comparing the solution to itself)
6:          End if
7:          If (solution j DOMINATES solution i):
8:              Increment N_i by 1
9:          End if
10:     End for
11:     If(N_i is 0):
            Add this solution (i'th solution) to the Front1
12:     End if
13: End for
14: print(Front1); prints the front 1 solutions
    (non dominated solutions) of the population.
15: Sort(Front1); sort according to any one objective
    (cycle time or energy) in ascending order.
16: Now the first and last solution in Front1 are the
    solutions with highest Crowding distance
17: Pick one solution from it randomly
    (considered to be the best solution in the population)
```

Moreover, all multi-objective studies use the concept of crowding distance, which sorts the front 1 solutions on a priority basis to select the best solution among them. This study, on the contrary, doesn't employ crowding distance but rather picks the best solution randomly from the corner solutions of the front 1 curve. This assumption holds true as the corner solutions hold high weightage criteria in the crowding distance method. All the above modifications significantly affect the computation and convergence of each algorithm. The newly adopted NDS approach correlates with the reduction of computational efforts and the modified discretization procedures mentioned in this section mostly aid in improved updation

methods, which in turn improves the convergence towards pareto optimal front.

6.1. Discrete Particle Swarm Optimization

Particle swarm optimization algorithm (PSO) is a population-based meta-heuristic algorithm perveived from swarm intelligence techniques and proposed by the ideology based on the herd mentality of animal societies (self organizing and division of labor), to collectively explore the best habitat amongst their population, considering several objectives and cautions. In PSO, a collection of particles together search for a solution by constantly conveying any superior solutions they find. Each population member then plans their consequent move from their respective individual finest discovery as well as the collective best achievement. There are two stoichiometric coefficient based equations, namely Velocity update and position update **(Rashid, 2016)**. Both the velocity and positional update equations define the relation between the current and successive position and velocity of each particle, respectively. The positional vector equation is given by:

$$X_i^{t+1} = X_i^t + V_i^{t+1} \quad (1)$$

Where, X_i^{t+1} is the position of any population in $(t+1)^{th}$ iteration, which is the immediate position after X_i^t (position of the member in $(t)^{th}$ iteration). The V_i^{t+1} denotes the velocity of the successive iteration, wherein the velocity update equation is:

$$V_i^{t+1} = (c1 * V_i^t) + (c2 * (P_{best,i} - X_i^t)) + (c3 * (G_{best} - X_i^t)) \quad (2)$$

From the randomly initialized swarm of task sequences that respects precedence relations, each swarm is decoded through workstation allocation procedures mentioned earlier. These decoded task sequences and process alternative sequences are calculated of their objectives and are then plotted to undergo NDS, in order to find the front 1. In every consecutive iteration, the best solution of each individual particle (called P_{best}) is updated as the positions of each particle gets updated as shown in **equation (1).** These positional updates are driven by the velocity update equation as shown in **equation (2),** which contains G_{best} & $P_{best,i}$ as parameters. The G_{best} is the best solution achieved by the entire swarm of particles at each iteration, and is determined by randomly picking one corner solution of the front 1 found using NDS as explained earlier. Therefore, the behavior of convergence is such that the positions and hence the P_{best} of individual particles are driven in the direction of the updated G_{best} in every iteration. The values found from the external archive set post termination of all iterations are considered to be the pareto optimal solutions. **Algorithm 4.1** shows the pseudocode steps in which the discrete PSO is implemented for the considered problem, with the aforementioned modifications.

Algorithm 4.1 Pseudo Code for PSO

1: Initialize **T;** termination criteria / number of iterations
2: Initialize **P;** population size (swarm has P particles)
3: Initialize a random **population** (task sequences and process alternatives)
4: Initialize **velocity vector** for the population
5: Initialize **p_best** same as the population
6: Using the decoding method, find cycle time and energy values for the population
7: **CT;** holds the cycle time of each particle in the population
8: **E;** holds energy consumption of each particle in the population
9: **EA;** external archive to store the pareto optimal front obtained
10: Using fast non-dominated sorting find front 1 solutions for the current population
11: Add all these solutions to the EA
12: From the found front 1 pick two corner points and assign one as **g_best** randomly
13: Initialize the coefficients (**c1, c2 & c3**)
14: **For** t=1 to T (termination criteria)
15: **For** i=1 to P (population size)
%UPDATE VELOCITY%
16: **Vi_t;** velocity of i'th particle
17: **Vi_t+1;** velocity of i'th particle
18: **p_best_i;** (p_best task sequence of i'th population)
19: **g_best;** (best task sequence in whole population)
20: **Vi_t+1;** found using **equation (2)**
%UPDATE TASK SEQUENCE%
21: **Xi_t;** task sequence of i'th particle
22: **Xi_t+1;** task sequence of i'th particle
23: **Xi_t+1;** found using **equation (1)**
%UPDATE PROCESS ALTERNATIVE%
24: **Yi_t;** process alternative of i'th particle
25: **Yi_t+1;** (process alternative of i'th particle)
p_best_pa_i; (process alternatives of i'th particle corresponding to p_best)
26: **g_best_pa_i;** (process alternative corresponding to g_best)
27: **Yi_t+1 = (Yi_t) + (p_best_pa_i) + (g_best_pa_i)**
%REPAIR UPDATED TASK SEQUENCE AND PROCESS ALTERNATIVE%
28: Modify Xi_t+1 so that the new sequence is discrete & follows precedence (as explained in **section 6.5**)
29: Modify Yi_t+1 such that the constraints are followed.
%UPDATE p_best%
30: Using the decoding method update cycle time and energy consumption for the i'th particle using Xi_t+1 and Yi_t+1

CT_i; (updated cycle time value for the i'th particle)
E_i; (updated energy rating for the i'th particle)
If (new CT_i & E_i dominate old p_best_i):
Update p_best_i
End if
End for
%UPDATE EXTERNAL ARCHIVE%
Using fast non-dominated sorting find front 1 for the current population
Add all these solutions to the EA
%UPDATE g_best%
From the found front 1 pick two corner points and assign one as g_best randomly
End for
%FINDING PARETO OPTIMAL FRONT%
Using fast non-dominated sorting for all the solutions in the external archive
Remove all solutions from EA that is not in front 1
print(EA); EA now holds the pareto optimal front solutions.

6.1.1 Improved discretization procedures for PSO:

From the existing discretization approaches that have been used by Rashid et al. **(Rashid, 2016)** for the various PSO operators, this study uses a simpler version of the same to discretize the developed algorithms throughout this paper. The pseudocodes of the simple and improved discretization procedures for PSO are given below for convenient reference for the reader.

Algorithm 4.1.1 Addition Operator 1
(Task Sequence + Velocity Vector)

V; velocity vector
X; task sequence vector
Y; vector to hold the solution after post the operation.
For i in range(1, Number of tasks):
    If(V[i] is X[i] or V[i] is 0):
        assign X[i] to Y[i]
    Else if(V[i] != 0):
        assign V[i] to Y[i]
    Else:
        assign 0 to Y[i]
    End if
End for

Algorithm 4.1.2 Subtraction Operator 1
(Task Sequence 1 - Task Sequence 2)

X1; task sequence 1
X2; task sequence 2
Y; vector to hold the solution after post the operation
For i in range(1, Number of tasks):
    If(X1[i] is X2[i]):
        assign X1[i] to Y[i]
    Else:
        assign X2[i] to Y[i]
    End if
End for

Algorithm 4.1.3 Multiplication Operator 1
(Co-efficient * Sequence Vector)

X; task sequence 1
C; co-efficient (c1 / c2 / c3)
Y; vector to hold the solution post the operation
For i in range(1, Number of tasks):
    R; generate random number between (0 to 1)
    If(R < C):
        assign X1[i] to Y[i]
    Else:
        assign 0 to Y[i]
    End if
End for

6.2. Discrete Teaching Learning-based Optimization

The Teaching Learning Based Optimization (TLBO) is a population-based meta-heuristic algorithm, perceived from the classroom environment of a teacher as well as interactions among students on the output of learners in a class. It has two phases, namely the Teaching phase and Learning phase. Through the teaching phase, learners get better from their teacher. The best population member here is the teacher in terms of both objectives and is used to bring learners (the remaining population) up to its level **(Rao, 2021)**. The teaching phase expression is given by:

$$X_i^{t+1} = X_i^t + (r * (X_{mean} - (T_f * X_{best}))) \quad (3)$$

Where, X_i^{t+1} denotes the sequence of any member in the $(t+1)^{th}$ iteration, which is the immediate sequence after X_i^t (sequence of the member in $(t)^{th}$ iteration). T_f is termed the teaching factor (1 or 2) and X_{mean} is the mean sequence of all population members of any iteration, whose method is explained in the upcoming sections. The learning phase expression is given by:

$$X_i^{t+1} = X_i^{t+1} + (r * (X_i^t (+ \text{ or } -) (T_p * X_p^t))) \quad (4)$$

Where, X_p denotes the partner population member picked randomly in order to undergo the learning phase and T_p is the partner factor.

Algorithm 4.2 shows the equations undergone by the population for both the phases. From the randomly initialized population of task sequences and process alternative sequences that satisfies precedence relations and constraints, each task sequence is decoded through workstation allocation procedures mentioned earlier. These decoded task sequences and process alternatives are calculated of their objectives and are then plotted to undergo NDS, in order to find front 1. In every iteration, each population member first undergoes the teaching phase as shown in **equation (3)**, where the X_{best} is the teacher randomly picked from one corner solution of the front 1 found using NDS as explained earlier. The teachers phase updates each member in a partially induced manner determined by T_f & r, and considers the collective position of every member determined by X_{mean}. The same population member then undergoes a learning phase as shown in **equation (4)**, where it is compared with a randomly picked different partner solution and updated partially according to T_p & r. Therefore, the methodology is such that each population learns from the best member, while communicating with each other to explore amongst them. They are ultimately driven in the direction of X_{best} in several iterations, where the external archive set values at the end of all iterations are considered to be the pareto optimal solutions. **Algorithm 4.2** shows the pseudocode steps in which the discrete TLBO is implemented for the considered problem, with the aforementioned modifications.

Algorithm 4.2 Pseudo code for TLBO

1: Initialize **T;** termination criteria / number of iterations
2: Initialize **P;** population size (class has P students)
3: Initialize a random **population** (task sequences and process alternatives)
4: Using the decoding method find cycle time and energy values for the population
5: **CT;** holds the cycle time of each student in population
6: **E;** holds energy rating of each student in population
7: **EA;** external archive to store the pareto optimal fronts
8: Using fast non-dominated sorting, find front 1 solutions for the current population.
9: Add all these solutions to the EA.
10: Initialize the coefficients (**Tf, Tp & r**)
11: **For** t=1 to T (termination criteria)
%CALCULATING MEAN OF POPULATION%
12: **X_mean;** holds mean of the population
13: Calculate X_mean using the method explained in **section 6.2.1.1**.
14: **For** i=1 to P (population size)
%TEACHING PHASE%
15: **X_best;** teacher for the class (holds the best solution of current population)
16: Using fast non-dominated sorting, find front 1 for the current population.
17: From the front 1 pick the two corner points and assign one point as X_best.
18: **Xi_t;** task sequence of i'th student
19: **Xi_t+1;** holds task sequence of i'th student.
20: **Xi_t+1 = Xi_t + (r * (X_mean - (Tf * X_best)))**
%LEARNING PHASE%
21: Generate a random number between [1 - population size].
22: Pick that solution as a partner solution for the current student.
23: **Xp_t;** partner solution (task sequence vector of p'th particle)
24: **If**(Xp_t better than Xi_t):
Xi_t+1 = Xi_t+1 + (r * (Xi_t - (Tp * Xp_t)))
25: **Else:**
Xi_t+1 = Xi_t+1 + (r * (Xi_t + (Tp * Xp_t)))
26: **End if**
%UPDATE PROCESS ALTERNATIVE%
27: **Yi_t;** process alternative of i'th student
28: **Yi_t+1;** process alternative of i'th student.
29: **X_best_pa;** process alternative corresponding to X_best.
30: **Yi_t+1 = Yi_t + X_best_pa**
%REPAIR UPDATED TASK SEQUENCE AND PROCESS ALTERNATIVE%
31: Modify Xi_t+1 such that the new sequence follows the precedence relation (as explained in **section 6.5**)
32: Modify Yi_t+1 such that the constraints are followed
%GREEDY SELECTION%
33: Keep track of the cycle time and energy consumption values of the i'th student before undertaking the teaching and learning phase as Ct_i_old and E_i_old.
34: Using the decoding method calculate new cycle time and energy consumption values for the i'th student using Xi_t+1 and Yi_t+1.
35: **CT_i_old;** cycle time value of i'th student.
36: **E_i_old;** energy consumption value of i'th student.
37: **CT_i_new;** new cycle time value for the i'th student.
38: **E_i_new;** new energy consumption value for the i'th student.
39: **If**(CT_i_new & E_i_new better than CT_i_old & E_i_old):
Update i'th student as Xi_t+1 & Yi_t+1
40: **Else:**
Don't update i'th student
41: **End if**
42: **End for**
%UPDATE EXTERNAL ARCHIVE%
43: Using fast non-dominated sorting find front 1 for the current population
44: Add all these solutions to the EA
45: **End for**
%FINDING PARETO OPTIMAL FRONT%
46: Using fast non-dominated sorting for all the solutions

in the external archive.
47: Remove all solutions from EA that is not in front 1
48: **print**(EA); EA now holds pareto optimal solutions.

6.2.1 Improved discretization procedures for TLBO:

Following the similar simple and improved discretization procedures used for PSO operators as shown in **section 6.1**, a modified discretization approach specifically suited to discretize the TLBO operators is carried out as shown in the following pseudocodes.

Algorithm 4.2.1 Mean Calculation
(Mean of All Task Sequences in the Population)

```
1:  Y; vector to hold the solution post the operation
2:  For i in range(1, Number of tasks):
3:        avg = 0; variable to calculate average
4:        Xj; task sequence [j'th member of the population]
5:        For j in range(1, population size):
6:                avg = avg + Xj[i]
7:        End for
8:        avg = avg / number of tasks
9:        If(avg is already present in Y):
10:             Do:
11:                R; generate random number between
                   [1 to number of tasks]
12:             While(R is already present in Y)
13:                Y[i] = assign R
14:       Else:
15:             Y[i] = avg
16:       End if
17: End for
```

Algorithm 4.2.2 Subtraction Operator 2
(Task Sequence 1 - (Co-efficient * Task Sequence 2))

```
1:  X1; task sequence 1
2:  X2; task sequence 2
3:  C; co-efficient (Tf / Tp/ r)
4:  Y; vector to hold the solution post the operation
5:  For i in range(1, Number of tasks):
6:        If(X1[i] is X2[i]): assign 0 to Y[i]
7:        Else:
              R; generate random number between (0 to 1)
8:            If(R < C): assign X2[i] to Y[i]
9:            Else: assign X1[i] to Y[i]
10:           End if
11:       End if
12: End for
```

Algorithm 4.2.3 Addition Operator 3
(Task Sequence1 + (Co-efficient * Task Sequence2))

```
1:  X1; task sequence 1
2:  X2; task sequence 2
3:  C; co-efficient (Tf / Tp/ r)
4:  Y; vector to hold the solution post the operation
5:  For i in range(1, Number of tasks):
6:        If(X1[i] is X2[i]):  assign X1[i] to Y[i]
7:        Else:
              R; generate random number between (0 to 1)
8:            If(R < C): assign X2[i] to Y[i]
9:            Else: assign 0 to Y[i]
10:           End if
11:       End if
12: End for
```

6.3. Discrete Migrating Birds Optimization

Migrating Birds Optimization algorithm derives its analogy from the flocking of birds in a V-shaped manner during their collective flight. The analogy is such that the V-shape is the whole population wherein the tip of the V is assumed to contain the leader (or) the best solution. This algorithm undergoes three phases, Leader improvement, Block improvement and Leader updation phases **(Li, 2021)**. **Algorithm 4.3** shows the different phases undergone by the population. From the randomly initialized population of task sequences and process alternative sequences that satisfies precedence relations and constraints, each task sequence and process alternative is decoded through workstation allocation procedures mentioned earlier. These decoded task sequences and process alternatives are calculated of their objectives and are then plotted to undergo NDS, in order to find front 1. The randomly picked corner solution from the front 1 curve serves as the leader of the V shape.

In the leader improvement phase, the initial leader improves itself by randomly generating 'k' number of neighbor solutions or task sequences cum alternatives and compares itself amongst them using the same NDS procedure. The best solution amongst the 'k+1' solutions is assigned the leader. Next, for the block improvement phase, each of the remaining population members improve themselves by randomly generating 'k-x' neighbor solutions or task sequences cum alternatives each, amongst which they are compared for their individual best solutions and updated using the NDS approach. Lastly, the leader is updated by comparing all the newly updated population members using NDS. This procedure is repeated until the termination criteria of the algorithm is met. MBO has the advantage of huge exploration capacity due to its widespread generation of intermediate neighborhood solutions, amongst which comparisons take place for the best solution. Thus, the tip of the V changes as soon as a new leader is discovered and after the termination criteria is met, all the leader solutions in the external archive are considered to be the pareto optimal

solutions set. The pseudocode of the algorithm below shows the specific steps in the phases involved.

Algorithm 4.3 Pseudo code for MBO

```
Initialize T; termination criteria / number of iterations
Initialize P; population size (population has P birds)
Initialize a random population
(task sequences and process alternatives)
Using the decoding method, find cycle time and
energy values for the population.
CT; holds the cycle time of each student
    in the population.
E; holds energy consumption of each student
    in the population
EA; external archive to store the pareto optimal
    front obtained
Using fast non-dominated sorting find
front 1 solutions for the current population
Add all these solutions to the EA
Initialize the coefficients (k & x)
For t=1 to T (termination criteria)
    %LEADER IMPROVEMENT OR
      REPLACEMENT PHASE%
    X_best; task sequence vector of the leader
            (holds the best solution of the population)
    Using fast non-dominated sorting find front 1
    for the current population
    From the found front 1 pick the two corner points
    and assign one point as X_best
    Generate k neighbor solution for this leader as:
    (k/3) by changing task sequence
    (k/3) by changing process alternative
    (k/3) by changing them both
    (explained in section 6.3.1)
    From these obtained k neighbor solutions,
    UPDATE the leader(X_best) with the solution
    which has the best cycle time and Energy values.
    %BLOCK IMPROVEMENT PHASE%
    For the remaining birds
    (left and right side of the flock):
    For i=2 to P (population size)
        X_it; task sequence vector of i'th bird
        Generate (k-x) neighbor solution for
        this i'th bird as:
        ((k-x)/3) by changing task sequence
        ((k-x)/3) by changing process
        ((k-x)/3) by changing them both
        (explained in section 6.3.1)
        From these obtained (k-x) neighbor
        solutions, UPDATE the i'th bird(X_i) with
        the solution which has best cycle time and
        Energy values
    End for
    %UPDATE EXTERNAL ARCHIVE%
    Using fast non-dominated sorting find front 1
    for the current population
    Add all these solutions to the EA
End for
%FINDING PARETO OPTIMAL FRONT%
Use fast non-dominated sorting for all the solutions
in the external archive
Remove all solutions from EA that is not in front 1
print(EA); EA now holds pareto optimal solutions.
```

6.3.1 Improved discretization procedures for MBO:

Following the similar simple and improved discretization procedures used for the operators of various algorithms considered previously, a modified discretization approach specifically suited to discretize the MBO operators is carried out as shown in the following pseudocodes.

Algorithm 4.3.1 Generation of neighborhood solutions 1 (By Changing Task Sequence Vector)

```
X; task sequence vector
Y; vector to hold the answer after carrying out the
operation
R; generate a random number between
[1 to number of tasks]
For i in range (1, R):
        Y[i] = X[i]
End for
For i in range (R+1, number of tasks):
    Do:
        q; generate random number between
        [1 to number of tasks]
    While(q is already present in Y)
        Y[i] = q
End for
Repair this task sequence Y
(as explained in section 6.5)
```

Algorithm 4.3.2 Generation of neighborhood solutions 2 (By Changing Process Alternative Sequence Vector)

1: To change and find neighbor solutions of a particular solution, swapping and single point mutation is performed such that the constraints are satisfied.
   - Swapping - randomly pick two indices from the sequence and swap their values such that the constraints are satisfied.
   - Single point mutation - randomly pick an index and change its value randomly such that the constraints are satisfied.

2: For example for a 4 station problem,
   Before changing - 1 3 1 2
   After performing swap operation - 1 3 2 1
   (last two positions are swapped)
   After performing single point mutation - 2 3 2 1
   (1st position is mutated)

6.4. Discrete Archimedes Optimization algorithm

The AOA or Archimedes optimization algorithm works on the basis of Archimedes principle, where buoyant force is exerted on objects partially immersed in water, to maintain their equilibrium. Assuming multiple objects immersed in the same fluid, where the objects are analogous to each population member and the fluid media is the search space, each object (population member) tries to reach equilibrium (optimized state). Each population member or task sequence cum process alternative has three values associated with it: Density, Volume and Acceleration **(Hashim, 2021)**. The three values are updated for each task sequence and process alternatives until the termination criteria is met as shown in **algorithm 4.4.** The best solution among the population is found initially using the NDS approach and is the one with best density, volume and acceleration associated with it. First, for each remaining population, density and volume are updated as the iteration starts using the respective updation equations shown in **equation (5) & equation (6)**. The algorithm contains two parameters T_f & d, being the transfer factor and density decreasing factor respectively, which increase gradually from 0 to 1 as they get updated in every iteration as shown in their respective updation equations **equation (7) & equation (8)**. The acceleration value followed by the task sequence cum process alternative of each population is then conditionally updated depending on the values of T_f and d as depicted in the below algorithm. These factors are based on the current and total iteration numbers, t and t_{max} respectively and play a vital role in bringing the whole population towards equilibrium, such that the algorithm has equal exploitation capability (when T_f <= 0.5, acceleration updation follows other random population (X_r) as shown in **equation (9)**) and exploitation capacity (when T_f > 0.5, acceleration updation follows current best solution (X_{best}) as shown in **equation (10)**). The updated acceleration values are normalized as given in **equation (11)** after which the population/sequence is updated using **equation (12) & equation (13)**.

$$Den_i^{t+1} = Den_i^t + r * (Den_{best} - Den_i^t) \quad (5)$$

$$Vol_i^{t+1} = Vol_i^t + r * (Vol_{best} - Vol_i^t) \quad (6)$$

$$T_f^t = \exp((t - t_{max}) / t_{max}) \quad (7)$$

$$d^t = \exp((t_{max} - t) / t_{max})) - (t / t_{max}) \quad (8)$$

$$Acc_i = ((Den_r + Vol_r * Acc_r) / (Den_i * Vol_i) \quad (\text{if } T_f <= 0.5) \quad (9)$$

$$Acc_i = ((Den_{best} + Vol_{best} * Acc_{best}) / (Den_i * Vol_i) \quad (\text{if } T_f > 0.5) \quad (10)$$

$$Acc_i(norm) = (0.9 * (Acc_i - Acc_{min}) / (Acc_{max} - Acc_{min})) + 0.1 \quad (11)$$

$$X_i^{t+1} = X_i^t + (d * Acc_i^{t+1} * (X_r^t - X_i^t)) \quad (\text{if } T_f <= 0.5) \quad (12)$$

$$X_i^{t+1} = X_i^t + (F * d * Acc_i^{t+1} * (X_{best} - X_i^t)) \quad (\text{if } T_f > 0.5) \quad (13)$$

For the specific purpose of solving and optimizing the HRC line balancing problem and its considerations in this study, the sequence updation procedures for the AOA algorithm are modified as opposed to the standard procedures introduced by Hashim et al **(Hashim, 2021)**. This is done so by removing the various randomization parameters that would otherwise contribute to excess computation. However, empirical trials were carried out to find the optimal task sequence before considering removing user-defined parameters. This gives an advantage to the AOA algorithm, that it requires no manual parameter fine tuning and the only parameters T_f and d are auto updated as the iterations progress. As discussed later in section 8**,** the relative performance of the hybridized AOA algorithm overrides the hybrids of the remaining algorithms. It also yields a large front while undergoing NDS, indicating that the algorithm does splendidly on the exploration end as well.

Moreover, as the density, volume and acceleration are all continuous variables, the aforementioned equations are simply applied for those variables. For updating the task sequence, the addition, subtraction and multiplication operators are the same as those given for the discretization procedures of TLBO in **section 6.2.1.**

Algorithm 4.4 Pseudo code for AOA

1: Initialize **T;** termination criteria / number of iterations
2: Initialize **P;** population size (population has P objects)
3: Initialize a random population
(task sequences and process alternatives)
4: Using the decoding method find cycle time and energy values for the population
5: **CT;** holds the cycle time of each student in the population
6: **E;** holds energy consumption of each student in the population
7: Initialize density, volume and acceleration for the population randomly
8: **D;** holds density value of each object in the population (ranges between 0 to 1)
9: **V;** holds volume value of each object in the population (ranges between 0 to 1)
10: **A;** holds acceleration value of each object in the population (ranges between 0 to 1)
11: **TF;** transfer operator
12: **d;** density decreasing factor
13: **EA;** external archive to store the pareto optimal front obtained
14: Using fast non-dominated sorting find front 1 solutions for the current population
15: Add all these solutions to the EA
16: **For** t=1 to T (termination criteria)

17: **For** i=1 to P (population size)
%UPDATE DENSITY & VOLUME%
18: **X_best;** best solution in the population
19: Using fast non-dominated sorting find front 1 for the current population
20: From the front 1 pick the two corner points and assign one point as X_best
21: **D_best;** holds density value corresponding to the X_best
22: **V_best;** holds acceleration value corresponding to the X_best
23: **Di_t;** density of i'th object in current iteration
24: **Vi_t;** volume of i'th object in current iteration
25: **Di_t+1;** holds density of i'th object for next iteration
26: **Vi_t+1;** holds volume of i'th object for next iteration
27: **r;** random number generated between (0 - 1)
28: **Di_t+1;** found using **equation (5)**
29: **Vi_t+1;** found using **equation (6)**
%UPDATE TRANSFER FACTOR AND DENSITY DECREASING FACTOR%
30: **TF;** calculated using **equation (7)**
31: **d;** calculated using **equation (8)**
%UPDATE ACCELERATION%
32: **Ai_t;** acceleration of i'th object in the current population
33: **Ai_t+1;** holds acceleration of i'th object for next iteration
34: **If**(TF <= 0.5):
Generate a random number r between (1 to population size)
35: **Ar_t;** acceleration of r'th object in the current iteration
36: **Dr_t;** density of r'th object in the current iteration
37: **Vr_t;** volume of r'th object in the current iteration
38: **Ai_t+1;** found using **equation (9)**
39: **Else:**
40: **A_best;** holds acceleration value corresponding to the X_best
41: **Ai_t+1;** found using **equation (10)**
42: **End if**
%NORMALIZE ACCELERATION%
43: **A_min;** minimum acceleration value in the population
44: **A_max;** maximum acceleration value in the population
45: **Ai_t+1(normalized);** found by **equation (11)**
%UPDATE TASK SEQUENCE%
46: **Xi_t;** task sequence of i'th object
47: **Xi_t+1;** holds task sequence of i'th object
48: **If**(TF <= 0.5):
49: **Xi_t+1;** found using **equation (12)**
50: **Else:** Generate a random number **F** (either -1 or +1)
51: **Xi_t+1;** found using **equation (13)**
52: **End if**
%UPDATE PROCESS ALTERNATIVE%
53: **Yi_t;** process alternative of i'th student
54: **Yi_t+1;** holds process alternative of i'th student
55: **X_best_pa;** process alternative corresponding to X_best
56: **Yi_t+1 = Yi_t + X_best_pa**
%REPAIR UPDATED TASK SEQUENCE AND PROCESS ALTERNATIVE%
57: Modify Xi_t+1 such that the new sequence follows the precedence relation **(**as explained in **section 6.5)**
58: Modify Yi_t+1 such that the constraints are followed
59: **End for**
%UPDATE EXTERNAL ARCHIVE%
60: Using fast non-dominated sorting find front 1 for the current population
61: Add all these solutions to the EA
62: **End for**
%FINDING PARETO OPTIMAL FRONT%
63: Using fast non-dominated sorting for all the solutions in the external archive
64: Remove all solutions from EA that is not in front 1
65: **print**(EA); EA now holds the pareto optimal solutions

6.5. Repair function for discretization procedures

In all the algorithms developed in this study, the updated task sequences cum process alternatives obtained after undergoing the proposed discretization methods in every iteration, might not follow the precedence relations. Sometimes, the task numbers might be repeated or some tasks might get missed out due to repetition of tasks. In order to repair these sequences, a repair function is proposed in this study and is implemented parallelly with the developed algorithms to make sure the ambiguities in yielding the desired task sequences are neglected and the updated population members always follow the precedence relation. An example of the differences in the ambiguous and the repaired task sequence is shown in **figure 10.** The methodology of the repair function is given as by the pseudocode below.

Algorithm 5 Pseudocode for Repair function method

1: **X;** task sequence vector to be repaired
2: **Y;** a vector to hold repaired task sequence
3: **Available;** a vector that has available tasks that can be allocated (follows precedence)
4: Insert tasks that have no precedence tasks into available vector
5: **n = 0;** keep track of index in vector X
6: **i = 0;** keep track of index in vector Y
7: **Do:**
8: **If**(n >= N[total number of tasks]):
9: Select a task randomly from the Available
10: T; randomly selected task from available
11: Insert T into Y[i]
12: Add tasks that have T as its precedence in Available

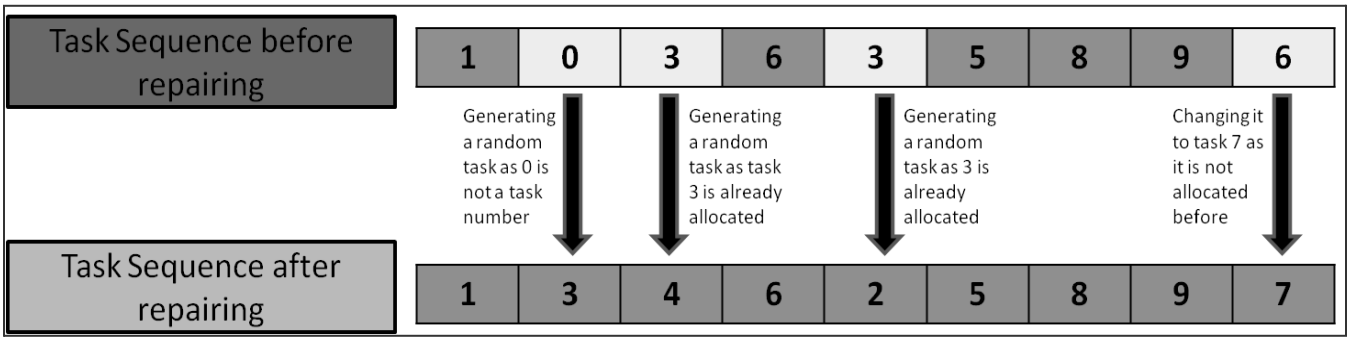

Figure 10. Working Illustration of the proposed repair function

```
            Remove T from the Available
            Increment i by 1
        Else if(X[n] in Available):
            Insert X[n] into Y[i]
            Add tasks that have X[n] as its precedence in Available
            Remove X[n] from the Available
            Increment n by 1
            Increment i by 1
        Else:
            Increment n by 1
        End if
        While(i < N [total number of tasks])
            Print(Y); Y contains repaired task sequence (that follows precedence relation)
```

7. Proposed Hybridization of discrete multi-objective algorithms

The domain of metaheuristic approaches are esteemed in dealing with a variety of complex and discrete optimization problems, especially when the decoding entities are substantially large and parameterized (NP-hard) **(Tansel, 2019)**. In order to continuously improve such a domain, hybridization procedures are among ways to combine the efficacy of phases involved in different popular metaheuristic algorithms. Apart from the competing and superior optimization mechanisms present in the four algorithms that are considered, this study proposes to hybridize and improve the individual performances of the four standard algorithms by embedding in them the Scout phase from the novel ABC algorithm.

In each iteration, the scout phase of the ABC keeps track of the individual population members or task cum process alternative sequences that are not improved/updated to a new objective value over the iterations **(Zeynel, 2020)**. **Algorithm 6** shows the sequential steps carried out, where a limit value is manually set for the allowed iteration count for which objective values (cycle time & energy consumption) of any population remains unchanged. Therefore, if the objective values of any population member remain unchanged over the iteration limit value that was set manually, this phase aids in the removal of that particular population member and replaces it with a random different population.

This mechanism of remote tracking, removal and replacement of members with repeating objective values prevents the algorithm from stagnating in local optima zones, enabling the population updation to converge better towards the narrowed set of pareto optimal values as each iterations progress. Thus, the result of this hybridization is the enhancement of the overall capacity to rapidly explore a bigger search space while increasing the chances of better exploitation (due to possible changes in recurring best solution). Moreover, the removed solution(s) may be the pareto optimal solution. For this purpose, they are stored/memorized separately in the EA using the elitist approach and are then compared at the algorithm termination point.

In the statistical results section of this paper, the comparative performance indications of the proposed hybridized algorithms and their standard equivalents are pointed out and discussed, to give a better perspective for the reader on the importance of hybridization approaches in general. The pseudocode for the considered Scout phase from the ABC algorithm is given below.

Algorithm 6 Pseudo code - Scout Phase of ABC Algorithm (Hybridization)

```
Counter; a vector that keep tracks of no. of times a solution in the population is unchanged
For i=1 to P (population size)
    Counter_i = 0; Counter for all the solutions in the population is set to 0 initially
End for
Limit; a value set by user
For t=1 to T (number of iterations)
    For i=1 to P (population size)
        CTi_old; holds cycle time value of i'th solution before updation
        Ei_old; holds energy value of i'th solution before updation
            …
        Run updation procedure using any algorithm
            …
```

11: **CTi_new;** holds cycle time value of i'th solution after updation
12: **Ei_new;** holds energy value of i'th solution after updation
13: **If**(CTi_old & Ei_old SAME AS CTi_new & Ei_new):
14: Increment Counter_i by 1
15: **End if**
16: **If**(Counter_i >= Limit):
17: Remove present i'th solution from the population
18: Add a new randomly generated solution in that position
19: Set Counter_i to 0
20: **End if**
21: **End for**
22: **End for**

8. Experimental Design and Results

This section presents the results obtained by analytical plots and statistical comparison of both the hybridized algorithms along with their standard models using three novel performance indications that depict the all-round characteristics of any metaheuristic algorithm. Further, the proposed simple fine tuning methodologies for each of the developed algorithms are given and their results as to the best fine-tuning criteria are discussed and displayed. To give the reader a better understanding of the differences in a fully automated (only Robotic) and human-robot (collaborative) assembly lines, the two configurations are compared and plotted. Lastly, the developed algorithms and the decoding methodology is supported by a case-study implementation in an electronic assembly industry.

8.1. Experimental Design

To address the lack of HRC worksetting datasets in literature, 32 standard HRC testing datasets as shown in **table 8** are categorized based on the sizes of tasks and workstations for decoding purposes. Of the 32 testing sets, tasks are categorized on the basis of their sizes: small (<50); medium (<100); large (>100), wherein the sizes of tasks increase gradually from 25 to 297, with varying workstation count. In this study, the approach followed in presenting the results of every algorithmic performance are based on these categories of task sizes, i.e., how each algorithm performs throughout these task size ranges and which algorithm has the highest significance at each task and workstation sizes of any HRC problem.

The detailed reference of the experimental setup that has been considered to carry out this study is given in the data generation methods under **section 4**. The quantity of manual workers aren't constrained whereas the study considered 3 types of robots with unlimited availability of each type. The energy consumption of robots largely varies depending on the application and thus the energy ratings of the robots were randomly generated in the range [0.2, 0.3]. The energy rating of humans translates relatively from wages and is empirically set as 0.1, which is lower than that of robots. The operational and other associated times are randomly generated as shown in the data generation methodology, using upper and lower bound values to account for deviations.

Table 8. Standard testing dataset premise used for comparison of algorithms

DATA SET no.	No. of Tasks	No. of stations	DATA SET no.	No. of Tasks	No. of stations
1	25	3	**17**	89	8
2	25	4	**18**	89	12
3	25	6	**19**	89	16
4	25	9	**20**	89	21
5	35	4	**21**	111	9
6	35	5	**22**	111	13
7	35	7	**23**	111	17
8	35	12	**24**	111	22
9	53	5	**25**	148	10
10	53	7	**26**	148	14
11	53	10	**27**	148	21
12	53	14	**28**	148	29
13	70	7	**29**	297	19
14	70	10	**30**	297	29
15	70	14	**31**	297	38
16	70	19	**32**	297	50

As in literature, the task times of cobots (worker + robot) are the lowest and the robotic times are the highest. The task constraint table in section 4 depicts the possibility of handling all tasks among every individual resource. The considerations for population size and iteration count (refer algorithm pseudocodes) were empirically determined and were taken as 30 each, for all developed standard & hybrid algorithms. The suitability of all the developed hybridized algorithms and their standard models are compared based on the task size categories and their individual significances amongst each other

regarding the various performance indicators. While running the code for each algorithm implementation, the generated data and decoding procedure was the same and all instances were solved for 5 repetitions each, to account for performance deviations if any. All the associated codes were written in C++, on the Dev C++ 6.5 platform. The codes of all the developed models and line implementation functions can be availed upon request.

8.2. Performance Indicators

Every new and improved metaheuristic model that is developed in various studies undergo performance comparisons using several indicators. The specific choice of indicators chosen depends on the developed algorithms and their comparison criteria. During the algorithmic generation of solutions in any iteration, the comparison of single-objective solutions involves merely a single expression whereas the comparison of multi-objective solutions involves a two-way conflicting approach **(Li, 2021)**. This leads to the necessity of comparing each solution in the consecutive iterations of their multi-objective domination criteria as explained in the algorithm pseudocodes and parallelly the performances of algorithms in yielding the desired solutions. For this study, four quantitative performance indicators are used, some of which are adopted from Rashid et al. **(Rashid, 2016)** to conduct the comparative studies:

1. ***Non-dominated solutions count*** (N), which denotes the number of non-dominated values found in the Pareto optimal set after the last iteration is complete. A higher value of (N) denotes better algorithmic performance.

2. ***Global error ratio*** (E_G), which denotes the percentage of the non-dominated solution count (N) to the global non-dominated value count (Pareto optimal set obtained from combining the individual pareto optimal sets of all algorithms under comparison). This ratio metric takes into consideration the relative non-dominated performance of any algorithm w.r.t the remaining algorithms. Hence, a lower value of (E_G) has an indirect relative correlation with (N) and signifies good performance of the algorithm.

3. ***Convergence metric*** (CM), which gives a measure of the extent of convergence towards the global Pareto optimal set. Here, the global Pareto optimal set is obtained from combining the individual pareto sets of all algorithms under comparison. It is mathematically determined by calculating the sum of euclidean distances (d_i) between the nearest end iteration values of non-dominated solutions and the global Pareto optimal set values and wholly dividing by N, as shown by **equation (14)**. Hence, a lower value of CM depicts better algorithmic performance.

$$CM = \frac{\sum_{i=1}^{N} d_i}{N} \tag{14}$$

4. ***Spacing metric*** (SM), which gives a measure of the relative distance between each solution in the end iteration values of the obtained non-dominated solutions, whose calculation is shown in **equation (15)** below. $\overline{d}$ denotes the average value of all relative distance (d_i). Hence, a lower spacing metric proves better performance wise.

$$SM = \sqrt{\frac{1}{N}\sum_{i=1}^{N} (d_i - \overline{d})^2} \tag{15}$$

5. ***Maximum spread*** (S_{max}), which measures the spread of the end iteration values from the non-dominated solutions obtained by any algorithm. As shown in **equation (16)**, the differences in the maximum and minimum values for the two objectives are considered at a time. A larger value of spread denoted better algorithmic performance.

$$S_{max} = \sqrt{\sum_{i=1}^{2} (min\, F_i - max\, F_i)^2} \tag{16}$$

8.3. Parametric Fine tuning

Every metaheuristic algorithm has certain parameters embedded in their updation equations, which aids to bias the equation in converging every solution towards an optimal value with different considerations. To aid in reducing the complexity of fine tuning of the algorithmic parameters, this study proposes a new range methodology to determine the best range for all the parameters involved in the four algorithms considered, based on task sizes. The different parameters involved can be seen from **equation (1)** to **equation (13)**. Depending on the influence of parameters in the update equations, the different parametric value ranges for all four algorithms are given as shown in **table 9**.

Using the performance indicators listed in **section 8.2**, the performance of each algorithm for all its associated six ranges are compared for the three categories of task sizes given in **section 8.1**. This way, the best parametric range is decided for each algorithm based on the size of tasks involved as shown in **table 10**.

Table 9. Proposed parameter fine tuning based on ranges

ALGORITHM	RANGES
PSO (**c1**-inertia weight **c2**-acceleration coefficient of P_best **c3**-acceleration coefficient of G_best)	• **Range 1 :** c1 ∈ (0, 0.5), c2+c3 <= 1.5 • **Range 2 :** c1 ∈ (0.5, 1), c2+c3 <= 1.5 • **Range 3 :** c2 ∈ (0, 0.5), c1+c3 <= 1.5 • **Range 4 :** c2 ∈ (0.5, 1), c1+c3 <= 1.5 • **Range 5 :** c3 ∈ (0, 0.5), c1+c2 <= 1.5 • **Range 6 :** c3 ∈ (0.5, 1), c1+c2 <= 1.5
TLBO (**r**-mean impact coefficient **Tf**-Teaching coefficient **Tp**-Learning coefficient)	• **Range 1 :** r ∈ (0, 0.5), Tf+Tp <= 1.5 • **Range 2 :** r ∈ (0.5, 1), Tf+Tp <= 1.5 • **Range 3 :** Tf ∈ (0, 0.5), r+Tp <= 1.5 • **Range 4 :** Tf ∈ (0.5, 1), r+Tp <= 1.5 • **Range 5 :** Tp ∈ (0, 0.5), r+Tf <= 1.5 • **Range 6 :** Tp ∈ (0.5, 1), r+Tf <= 1.5
MBO (**K** - number of neighborhood solutions of current leader **X** - represents any number less than K)	• **Range 1 :** K=2, X=1 • **Range 2 :** K=3, X=1 • **Range 3 :** K=5, X=2 • **Range 4 :** K=5, X=3 • **Range 5 :** K=7, X=3 • **Range 6 :** K=10, X=5
AOA	No fine tuning needed as parameters are neglected for the developed & improved AOA.

Table 10. Conclusive result of fine tuning of parameters

DATA SIZE	PSO	TLBO	MBO	AOA
SMALL	c1 = [0 - 0.5] c2 + c3 <= 1.5	Tf = [0 - 0.5] r + Tp <= 1.5	K = 5 X = 3	-------
MEDIUM	c3 = [0.5 - 1] c1 + c2 <= 1.5	Tp = [0.5 - 1] r + Tf <= 1.5	K = 3 X = 1	-------
LARGE	c2 = [0.5 - 1] c1 + c3 <= 1.5	Tf = [0 - 0.5] r + Tp <= 1.5	K = 5 X = 2	-------

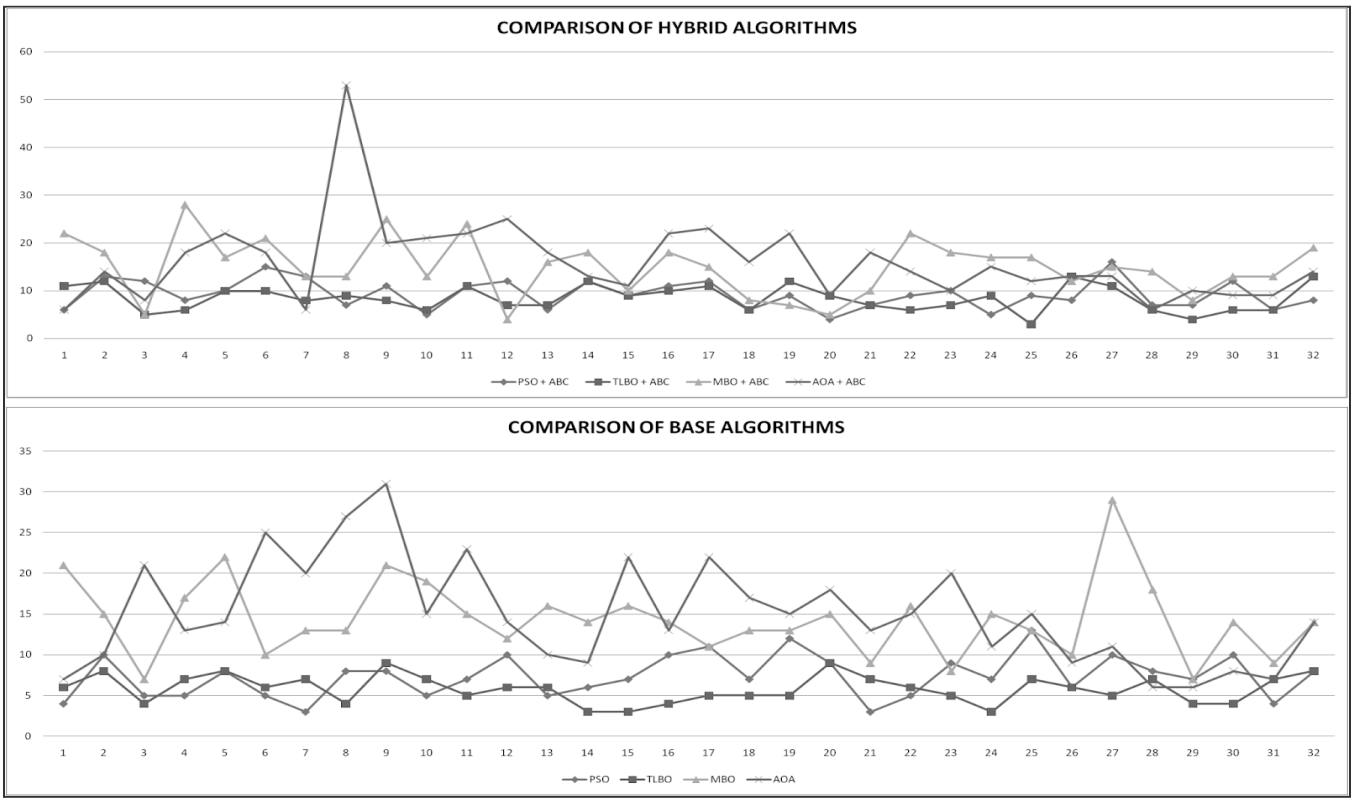

Figure 11. Line plot of Performance Indicator 1 (N)

8.2. Comparative Results

The comparative studies presented in this paper primarily focus on comparing the developed hybrid algorithms (PSO+ABC, TLBO+ABC, MBO+ABC, AOA+ABC) against their standard models. For the purpose of deciding the suitability among the four algorithms for line balancing studies, comparisons have been extended such that the hybridized and standard models are compared within themselves as well as in a collective fashion to test for inter-algorithm significance in performances. For instance, **figure 11** shows the comparison plots of both standard & hybridized versions of PSO, TLBO, MBO and AOA using the performance indicator 1 (N) (on y-axis) for the 32 developed standard testing HRC datasets (on x-axis). In this chapter, only the first three performance indicators, i.e., N, E_G & CM are considered sufficient as they hold high priority in determining overall algorithmic performances and excess computation can be avoided. It can be observed that AOA shows higher peaks indicating a greater performance compared to the rest. With reference to **section 8.2**, similar plots are generated for the various performance indicators as shown in **figure 12** and **figure 13**. The AOA algorithm manifests a better performance with a lower error ratio and convergence metric. The reader could however compare any relative algorithmic performances based on the different task sizes by simply navigating through the abscissa (task sizes) and ordinate (performance values) of the plots.

Apart from the hybridized & standard models of both AOA and MBO, all other algorithms have almost no spread amongst them in the plots, signifying that their performances throughout the HRC dataset environments are more or less the same. To get numerical estimates and confirmation on the performance significations that match the observation from plots, statistical studies were carried out using Friedman's test as shown in **table 11** and **table 12**. The test confirms significance for ⅔ priority indicators (p-value < 0.05), which is evident from the spread of the line plots shown.

Given the existence of significance for two of the prior performance indicators (N) and (E_G), Post-hoc testing is implemented for comparing the relative performances of all individual algorithms, to test their significance in the two indicators. From the post-hoc results carried out as shown in **table 13** for indicator (N), it could be seen that the AOA algorithm and its hybrid shows the most significance when compared with the rest of the algorithms. The underlined treatment pair cells shown in the results table depict the comparisons that involve AOA. As this study categorizes the 32 task datasets into sizes of small, medium and large, it can also be concluded that the standard and hybridized AOA algorithm performs better than the rest of the algorithms for all task sizes. Similarly, **table 14** shows the post-hoc test results carried out for indicator (E_G), which furthermore shows the highest significance count of the hybridized AOA algorithm. To extend the grounds of algorithmic performance, the pareto optimal fronts obtained through NDS by all the developed algorithms after the termination criteria of 30 iterations are plotted to visualize the relative and collective positions of the multi-objective optimized solutions.

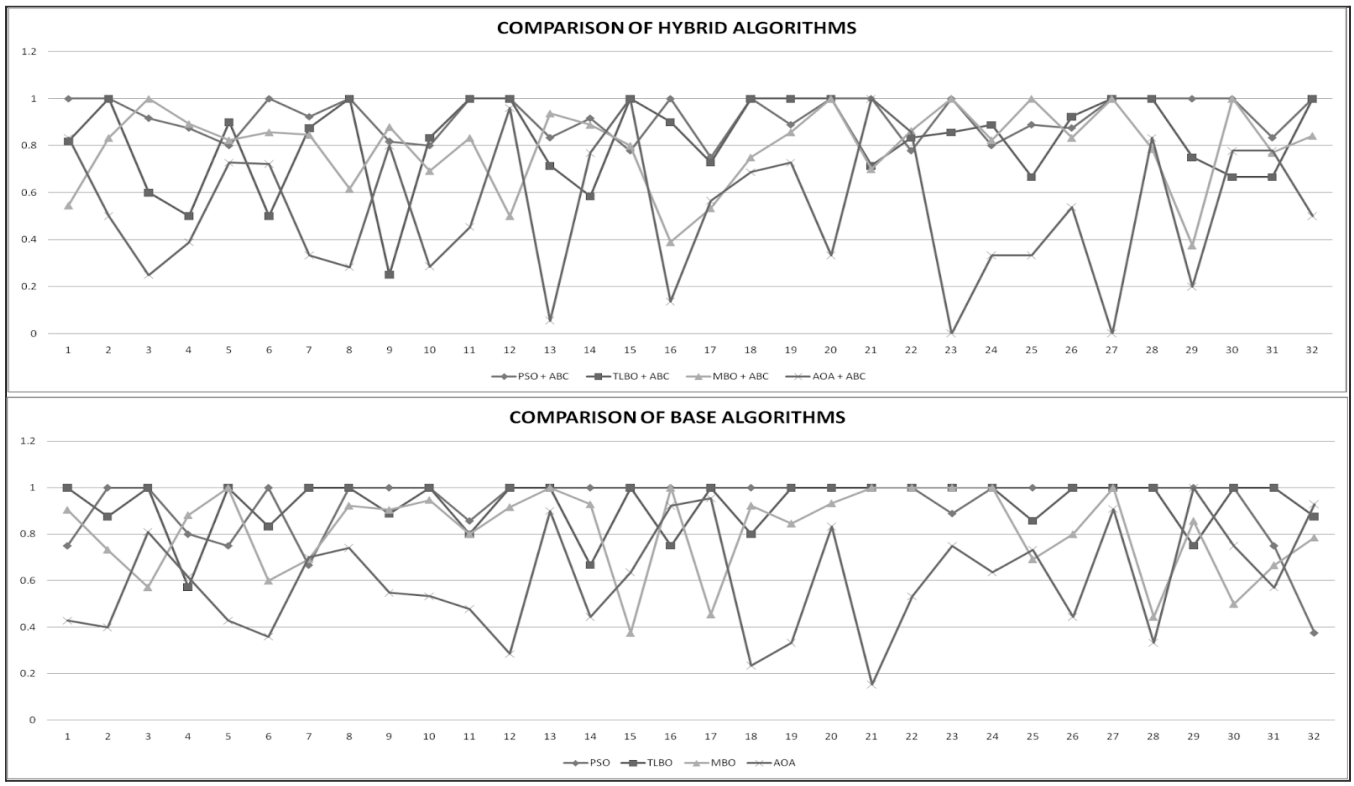

Figure 12. Line plot of Performance Indicator 2 (E_G)

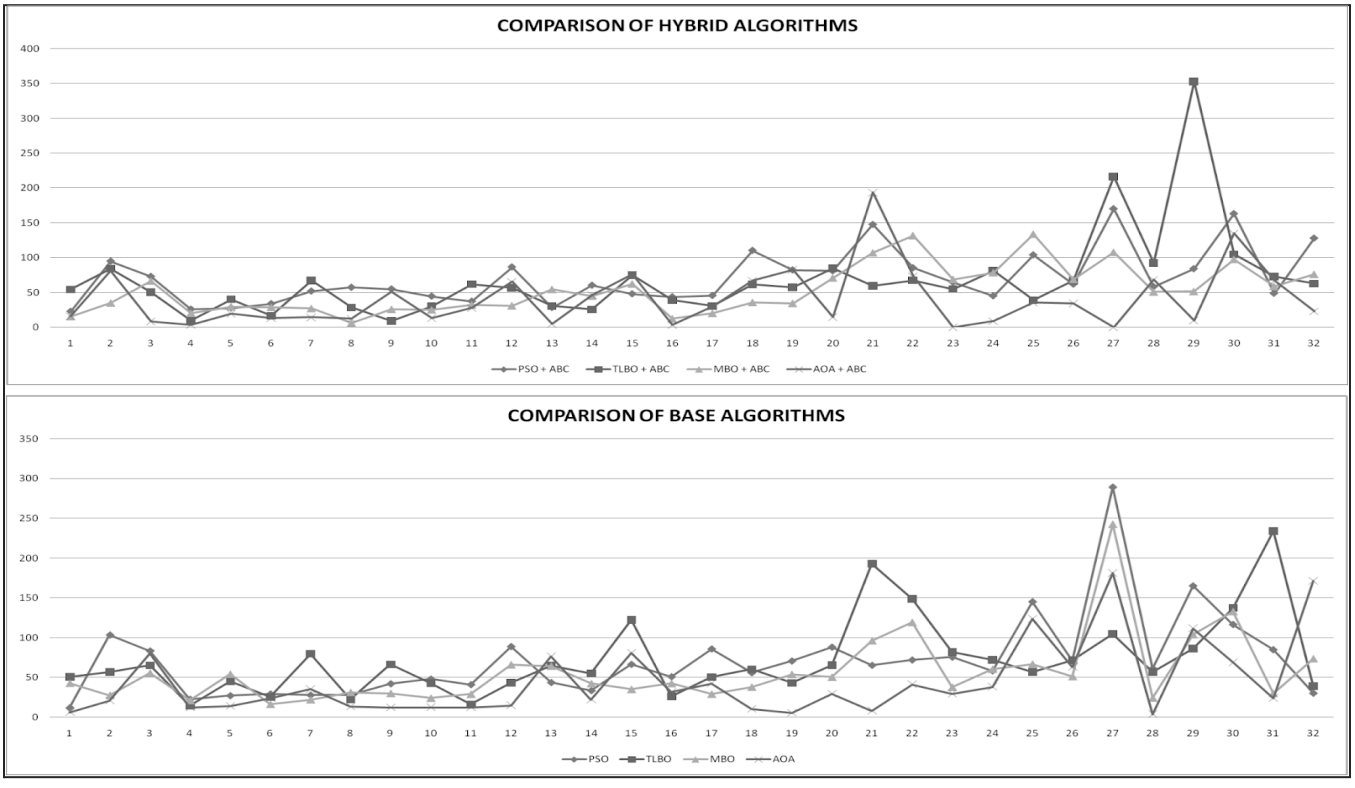

Figure 13. Line plot of Performance Indicator 3 (CM)

Table 11. Friedman Test results for standard algorithms

PERFORMANCE INDICATOR	Number of data sets **(n)**	Chi-square value **(χ2)**	Degrees of freedom **(df)**	**p- value** (significance indicator)
Non dominated solutions count (N)	32	61.434	3	< 0.00001
Global error ration (E_G)	32	29.025	3	0.00028
Convergence metric (CM)	32	23.888	3	0.287

Table 12. Friedman Test results for hybrid algorithms

PERFORMANCE INDICATOR	Number of data sets **(n)**	Chi-square value **(χ2)**	Degrees of freedom **(df)**	**p- value** (significance indicator)
Non dominated solutions count (N)	32	39.722	3	< 0.00001
Global error ration (E_G)	32	18.35	3	0.00037
Convergence metric (CM)	32	18.488	3	0.343

Table 13. Scheffé multiple comparison test results using non-dominated solutions count(N)

Treatments pair	**Scheffé T-statistic**	**Scheffé p-value**	**Scheffé inference**
PSO vs PSO+ABC	1.5206	0.9397136	insignificant
PSO vs TLBO	1.2215	0.9822245	insignificant
PSO vs TLBO+ABC	0.8725	0.9977599	insignificant
PSO vs MBO	5.5839	0.0001095	** p<0.01
PSO vs MBO+ABC	6.0575	1.3529e-05	** p<0.01
PSO vs AOA	6.1323	9.5683e-06	** p<0.01
PSO vs AOA+ABC	6.8552	2.7179e-07	** p<0.01
PSO+ABC vs TLBO	2.7421	0.3805235	insignificant
PSO+ABC vs TLBO+ABC	0.6481	0.9996801	insignificant
PSO+ABC vs MBO	4.0633	0.0238375	* p<0.05
PSO+ABC vs MBO+ABC	4.5369	0.0056326	** p<0.01
PSO+ABC vs AOA	4.6117	0.0043954	** p<0.01
PSO+ABC vs AOA+ABC	5.3346	0.0003057	** p<0.01
TLBO vs TLBO+ABC	2.0939	0.7339294	insignificant

TLBO vs MBO	6.8053	3.5161e-07	** p<0.01
TLBO vs MBO+ABC	7.2790	2.8504e-08	** p<0.01
TLBO vs AOA	7.3537	1.8919e-08	** p<0.01
TLBO vs AOA+ABC	8.0766	3.0302e-10	** p<0.01
TLBO+ABC vs MBO	4.7114	0.0031316	** p<0.01
TLBO+ABC vs MBO+ABC	5.1850	0.0005516	** p<0.01
TLBO+ABC vs AOA	5.2598	0.0004116	** p<0.01
TLBO+ABC vs AOA+ABC	5.9827	1.9047e-05	** p<0.01
MBO vs MBO+ABC	0.4736	0.9999615	insignificant
MBO vs AOA	0.5484	0.9998958	insignificant
MBO vs AOA+ABC	1.2713	0.9775765	insignificant
MBO+ABC vs AOA	0.0748	1.0000000	insignificant
MBO+ABC vs AOA+ABC	0.7977	0.9987435	insignificant
AOA vs AOA+ABC	0.7229	0.9993403	insignificant

Table 14. Scheffé multiple comparison test results using global error ratio(E_G)

Treatments pair	**Scheffé T-statistic**	**Scheffé p-value**	**Scheffé inference**
PSO vs PSO+ABC	0.2398	0.9999996	insignificant
PSO vs TLBO	0.1123	1.0000000	insignificant
PSO vs TLBO+ABC	2.4252	0.5548421	insignificant
PSO vs MBO	2.4807	0.5235409	insignificant
PSO vs MBO+ABC	2.8873	0.3084131	insignificant
PSO vs AOA	6.9436	1.7142e-07	** p<0.01
PSO vs AOA+ABC	8.5068	2.2587e-11	** p<0.01
PSO+ABC vs TLBO	0.1275	1.0000000	insignificant
PSO+ABC vs TLBO+ABC	2.1854	0.6870097	insignificant
PSO+ABC vs MBO	2.2409	0.6573650	insignificant
PSO+ABC vs MBO+ABC	2.6475	0.4308202	insignificant

PSO+ABC vs AOA	6.7038	5.9074e-07	** p<0.01
PSO+ABC vs AOA+ABC	8.2670	9.7203e-11	** p<0.01
TLBO vs TLBO+ABC	2.3130	0.6178117	insignificant
TLBO vs MBO	2.3684	0.5868431	insignificant
TLBO vs MBO+ABC	2.7750	0.3635503	insignificant
TLBO vs AOA	6.8314	3.0742e-07	** p<0.01
TLBO vs AOA+ABC	8.3945	4.4892e-11	** p<0.01
TLBO+ABC vs MBO	0.0554	1.0000000	insignificant
TLBO+ABC vs MBO+ABC	0.4621	0.9999675	insignificant
TLBO+ABC vs AOA	4.5184	0.0059837	** p<0.01
TLBO+ABC vs AOA+ABC	6.0815	1.2109e-05	** p<0.01
MBO vs MBO+ABC	0.4066	0.9999865	insignificant
MBO vs AOA	4.4629	0.0071595	** p<0.01
MBO vs AOA+ABC	6.0261	1.5628e-05	** p<0.01
MBO+ABC vs AOA	4.0563	0.0243050	* p<0.05
MBO+ABC vs AOA+ABC	5.6194	9.4186e-05	** p<0.01
AOA vs AOA+ABC	1.5631	0.9303866	insignificant

From **figure 14,** the pareto fronts obtained by the hybridized AOA algorithm yields a relatively more spread front signifying the increased exploration capacity due to addition of the scout phase mechanism. The hybrid AOA front is also the closest to the origin of the multi-objective space, signifying minimized/optimized dual-objective task sequence/process alternative (population member) and enhanced exploitation capacity of the algorithm.

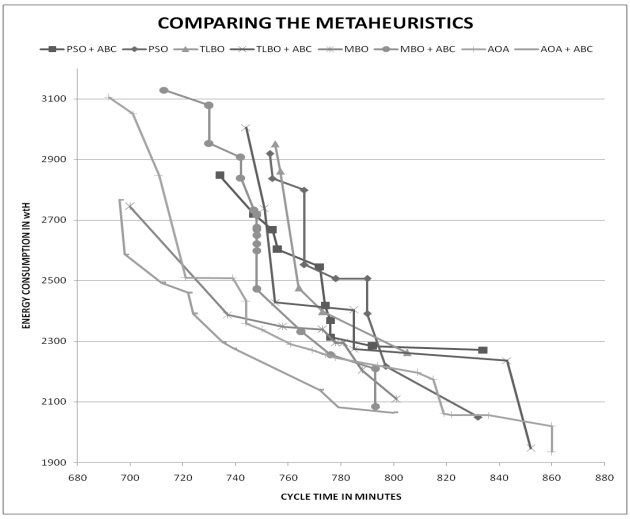

Figure 14. End iteration Pareto optimal fronts obtained by all 8 algorithms

8.2. Case study:

With reference to the set of hybridized algorithms developed in this study along with the different data generation methods and flexible resource constraint allocation capabilities, the discussed decoding methodology of HRC resources for efficient line balancing are supported by a case study done in an electronics assembly industry involving semi-automated assembly of PCB boards and components of complex power housing units. This study is carried out to also validate the performance of the developed metaheuristic algorithms by considering a suitable sub-assembly process (DTDC Housing Assembly).

8.2.1. Line description & modeling:

The current scenario of the considered sub-assembly process are as follows: All tasks in the observed sub-assembly segment were carried out manually (study aims to semi-automate existing manual operations). The workstation count is subjected to shop-floor space availability and was considered to be from 2 to 5. The housing assembly consisted of 23 tasks pertaining to task characteristics like shifting, placing & positioning, fastening, intermediate torque, quality & part inspections, and intricate wire positioning (only done manually). Before testing the HRC line balancing performance by the developed metaheuristics, the primary objective involved converting the existing manual sub-assembly process into a "semi-automated" one by considering managerial cost and automation resource implications.

The methodology carried out to model such a collaborative assembly system are as follows:

- All tasks are observed and documented of their operation times, their possible implementation methods and task precedence relations.
- The best possible automation to be introduced is discussed with the IE & automation teams of the industry for their specifications and suitability in assembling sensitive electronic components. After careful analysis of all the recorded tasks, it was decided that two types of robots, one with mechanical grippers to perform pick & place operations and another with end-effectors of suction tube for delicate portions.
- The task constraints of all the available resources were formulated after ergonomic and safety considerations.
- From various industrial robot specification sources, the average energy ratings of both the robotic types (6-axis-7kg(max)-articulated automatic numerical control robots) are taken to be in the range of [0.6-0.8] kW.
- The setup times measured for both types of robots for all tasks were observed to be very insignificant w.r.t the operational times. Hence, it can be assumed for this study that the operational time bounds would have accounted for the individual setup times.
- Since the assembly is highly compact and integrated, the resource allocations are optimally spread to avoid the possibility of collisions.
- The availability of human workers is assumed unconstrained.

The modeled HRC sub-assembly inputs of 23 tasks finally need to be optimally allocated among the possible number of workstations by assigning the available manual and given collaborative robot resources with the aim to balance the line by reducing total cycle time & line energy consumption. It should be noted that due to confidential reasons inline with the company's legal policies, only the outline data of the sub-assembly process has been listed.

8.2.2. Performance analysis and results discussion:

After running the hybridized AOA algorithm for the given modeled data, the resulting cycle time and energy consumption values for different workstation counts are first compared between the existing manual assembly line to that of the generated semi-automated line. Thus, the comparison shown in **figure 6** provides the necessary insights for deciding the right workstation count by comparing the simultaneous relative decrease in cycle times and increase in energy consumption for every workstation count (represented in the abscissa). Through the multi-objective decoding implementations carried out, a workstation count of '3' shows the largest decrease in cycle time observed for a relatively less hike in line energy consumption. Although it is certain that energy consumption is increased when automation is added, the long term productivity is increased which compensates for the relatively small hike in energy consumption.

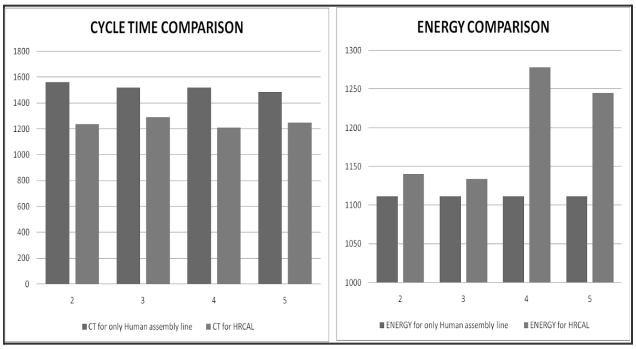

Figure 15. Insights from case-study

Having chosen three workstations as the optimal shop-floor space utilization criteria, the hybrid AOA algorithm is run for the said workstation count and the

input models depicted earlier. **Figure 19** shows the HRC allocation that has been decoded by the algorithm with the best task sequence solution among the generated pareto optimal set. According to this allocation, the first workstation is assigned a manual space, the second with a collaborative space of robot R1, and the third with yet another collaborative space of robot R2, all with their best possible respective tasks. **Figure 18** shows the only-manual process that is currently carried out, with clear indications of reduced objectives as compared to the HRC allocation. Through this case-study implementation, the use of metaheuristics in solving line balancing problems proves to be largely flexible in arriving at the best optimal sequences, considering several resource and management constraints. In practical situations, high volume projects in an industry site the need to automate most of their processes effectively. With the available level of current automation and the long term anticipated growth of unfolding multiple projects, managers can extend such HRC studies into manual lines that are critical to the throughput of a product.

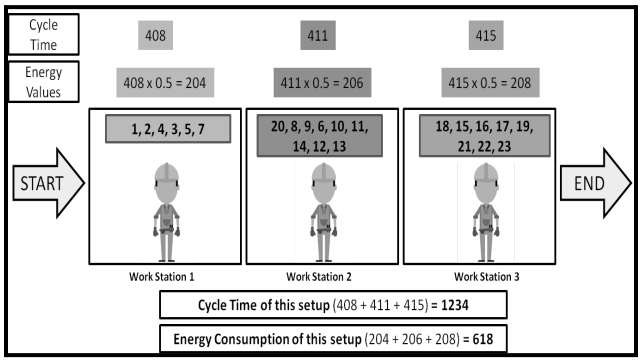

Figure 16. Optimal manual allocation (current) generated

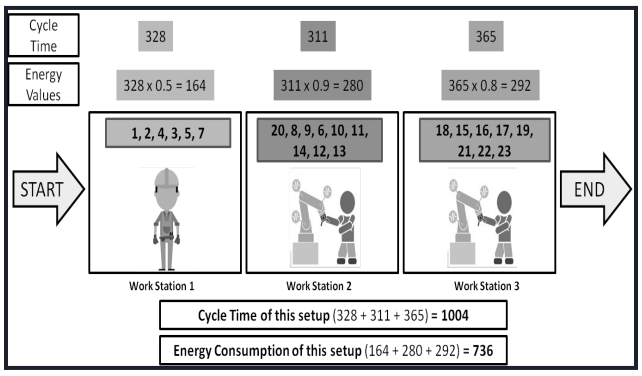

Figure 17. Optimal HRC allocation generated based on the collaborative model

9. Conclusions and future directions:

The developed algorithms altogether were used in decoding to an optimal assembly line sequence from the generated data considering task times, setup and sequence dependent setup times, resource availability counts and specifications, and allocation constraints proposed by considering safety, ergonomic and economic factors. The proposed fine tuning methodologies and discretization procedures in this study for all the algorithms can be aligned as a benchmark towards further easy implementation of future studies that employ similar metaheuristics to solve discrete problems.

Having identified the lack of standard testing HRC datasets, a set of 32 datasets are formulated of small, medium and large task sizes for testing any developed algorithm for their performances when task sizes of the test cases are of different sizes. Moreover, when these developed standard datasets are used to compare significance in performance indication for the developed algorithms, it helped to prove that 2 out of 3 considered performance indicators actually showed better performances as shown in **table 5**. The statistical tests' performances gave another insight that the recently proposed hybridized and standard AOA algorithm showed more than 50% better performance when compared to the others. This test highlights the potential of AOA in being used as an ideal metaheuristic for solving discrete ALB and assembly sequence planning, which can also be supported by the large and near-to-origin optimal pareto fronts obtained for AOA. Throughout the study, the pseudocodes of every proposed framework are sequentially explained to save time in interpretation for new researchers. The study also sheds light on task sequence repair functions, whose associated problems are usually faced by researchers in coding discrete applications.

Though the study of metaheuristic performances and line balancing for HRC compliances considers a lot of assumptions, these instances would prove particularly useful when applied for a real-world formulated case study in an appropriate industry. The primary objective of conducting case studies can be the research on introducing a certain amount of automation for the current traditional methods involved, if any, for any industry. The line balancing results obtained in this study for the manual turned semi-automated electronic sub-assembly process showed the desired results of lowering the cycle time with considerable energy consumption of automation resources, as explained in **section 3.6**. It also proves the viability of using metaheuristic algorithms, developing their performances and then running a vast multitude of constraint induced task sequence, times and alternatives data to produce the best set of optimal assembly line allocation results.

For research on HRC line balancing in the coming times, many concepts related to production planning and control methods could be adopted. For instance, the layout alternatives for different HRC settings as listed briefly by Qiuhua et al. **[24]** can be looked at much deeper. As depicted in this study, the developed significant algorithms and decoding structures can be implemented in real-case industrial problems as most industries provide the flexibility of several product assembly lines and the research on HRC could be aligned with real world industrial considerations rather than pure assumptions.

REFERENCES

[1] Ahmad Taheri, Keyvan RahimiZadeh, Ravipudi Venkata Rao. An efficient Balanced Teaching-Learning-Based optimization algorithm with Individual restarting strategy for solving global optimization problems, Information Sciences, Volume 576, 2021, Pages 68-104, ISSN 0020-0255, https://doi.org/10.1016/j.ins.2021.06.064

[2] Amir Nourmohammadi, Masood Fathi, Amos H.C. Ng, Balancing and scheduling assembly lines with human-robot collaboration tasks, Computers & Operations Research, Volume 140, 2022, 105674, ISSN 0305-0548, https://doi.org/10.1016/j.cor.2021.105674

[3] Ana Correia Simões, Ana Pinto, Joana Santos, Sofia Pinheiro, David Romero, Designing human-robot collaboration (HRC) workspaces in industrial settings: A systematic literature review, Journal of Manufacturing Systems, Volume 62, 2022, Pages 28-43, ISSN 0278-6125, https://doi.org/10.1016/j.jmsy.2021.11.007

[4] Christian Weckenborg, Thomas S. Spengler, Assembly Line Balancing with Collaborative Robots under consideration of Ergonomics: a cost-oriented approach, IFAC-PapersOnLine, Volume 52, Issue 13, 2019, Pages 1860-1865, ISSN 2405-8963, https://doi.org/10.1016/j.ifacol.2019.11.473

[5] Chutima, Parames. (2022). A comprehensive review of robotic assembly line balancing problem. Journal of Intelligent Manufacturing. 33. 10.1007/s10845-020-01641-7

[6] Dalle Mura, M., Dini, G. Job rotation and human–robot collaboration for enhancing ergonomics in assembly lines by a genetic algorithm. *Int J Adv Manuf Technol* 118, 2901–2914 (2022). https://doi.org/10.1007/s00170-021-08068-1

[7] Deb K. Multi-objective optimization using evolutionary algorithms. Chichester: John Wiley & Sons, 2002

[8] Deb K, Pratap A, Agarwal S, Meyarivan T (2002) A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans Evol Comput 6(2):182–197

[9] Gualtieri, L., Rauch, E. & Vidoni, R. Methodology for the definition of the optimal assembly cycle and calculation of the optimized assembly cycle time in human-robot collaborative assembly. *Int J Adv Manuf Technol* 113, 2369–2384 (2021). https://doi.org/10.1007/s00170-021-06653-y

[10] Hamzadayi, A. (2018). Balancing of mixed-model two-sided assembly lines using teaching-learning based optimization algorithm. *Pamukkale University Journal of Engineering Sciences, 24*, 682-691

[11] Hashim, F.A., Hussain, K., Houssein, E.H. *et al.* Archimedes optimization algorithm: a new metaheuristic algorithm for solving optimization problems. *Appl Intell* 51, 1531–1551 (2021). https://doi.org/10.1007/s10489-020-01893-z

[12] Kashani, A.R., Chiong, R., Mirjalili, S. *et al.* Particle Swarm Optimization Variants for Solving Geotechnical Problems: Review and Comparative Analysis. Arch Computat Methods Eng 28, 1871–1927 (2021). https://doi.org/10.1007/s11831-020-09442-0

[13] Kathryn E. Stecke & Mahdi Mokhtarzadeh (2022) Balancing collaborative
human–robot assembly lines to optimise cycle time and ergonomic risk, International Journal of Production Research, 60:1, 25-47, DOI:10.1080/00207543.2021.1989077

[14] Leonardo Borba, Marcus Ritt, Cristóbal Miralles, Exact and heuristic methods for solving the Robotic Assembly Line Balancing Problem, European Journal of Operational Research, Volume 270, Issue 1, 2018, Pages 146-156, ISSN 0377-2217, https://doi.org/10.1016/j.ejor.2018.03.011

[15] Li, Zixiang & Janardhanan, Mukund Nilakantan & S.G., Ponnambalam. (2021). Cost-oriented robotic assembly line balancing problem with setup times: multi-objective algorithms. Journal of Intelligent Manufacturing. 32. 10.1007/s10845-020-01598-7.

[16] Li, Z., Janardhanan, M.N. & Tang, Q. Multi-objective migrating bird optimization algorithm for cost-oriented assembly line balancing problem with collaborative robots. *Neural Comput & Applic* 33, 8575–8596 (2021). https://doi.org/10.1007/s00521-020-05610-2

[17] Malik, A.A. and Bilberg, A. (2019), "Complexity-based task allocation in human-robot collaborative assembly", *Industrial Robot*, Vol. 46 No. 4, pp. 471-480. https://doi.org/10.1108/IR-11-2018-0231

[18] Michela Dalle Mura, Gino Dini, Designing assembly lines with humans and collaborative robots: A genetic approach, CIRP Annals, Volume 68, Issue 1, 2019, Pages 1-4, ISSN 0007-8506, https://doi.org/10.1016/j.cirp.2019.04.006

[19] Miguel Vieira, Samuel Moniz, Bruno S. Gonçalves, Tânia Pinto-Varela, Ana Paula Barbosa-Póvoa & Pedro Neto (2021): A two-level optimisation-simulation method for production planning and scheduling: the industrial case of a human–robot collaborative assembly line, International Journal of Production Research, DOI:10.1080/00207543.2021.1906461

[20] Mohd Fadzil Faisae Ab Rashid, Windo Hutabarat, Ashutosh Tiwari. (2016). Multi-objective discrete particle swarm optimisation algorithm for integrated assembly sequence planning and assembly line balancing. DOI: 10.1177/0954405416673095

[21] N. Boysen, P. Schulze and A. Scholl, Assembly line balancing: What happened in the last fifteen years? European Journal of Operational Research, https://doi.org/10.1016/j.ejor.2021.11.043

[22] Nicole Berx, Wilm Decré, Ido Morag, Peter Chemweno, Liliane Pintelon, Identification and classification of risk factors for human-robot collaboration from a system-wide perspective, Computers & Industrial Engineering, Volume 163, 2022, 107827, ISSN 0360-8352, https://doi.org/10.1016/j.cie.2021.107827

[23] Parsopoulos, Konstantinos & Vrahatis, Michael. (2008). Multi-objective particle swarm optimization approaches. 10.13140/2.1.5189.4721

[24] Qiuhua Tang, Zixiang Li, LiPing Zhang, and Chaoyong Zhang. 2017. Balancing stochastic two-sided assembly line with multiple constraints using hybrid teaching-learning-based optimization algorithm. Comput. Oper. Res. 82, C (June 2017), 102–113. DOI:https://doi.org/10.1016/j.cor.2017.01.015

[25] Tamás Koltai, Imre Dimény, Viola Gallina, Alexander Gaal, Chiara Sepe, An analysis of task assignment and cycle times when robots are added to human-operated assembly lines, using mathematical programming models, International Journal of Production Economics, Volume 242, 2021, 108292, ISSN 0925-5273, https://doi.org/10.1016/j.ijpe.2021.108292

[26] Tansel Dokeroglu, Ender Sevinc, Tayfun Kucukyilmaz, Ahmet Cosar, A survey on new generation metaheuristic algorithms, Computers & Industrial Engineering, Volume 137, 2019, 106040, ISSN 0360-8352, https://doi.org/10.1016/j.cie.2019.106040

[27] Weckenborg, C., Kieckhäfer, K., Müller, C. *et al.* Balancing of assembly lines with collaborative robots. *Bus Res* 13, 93–132 (2020). https://doi.org/10.1007/s40-0685-019101-y

[28] Y. Ma, X. Zhang, J. Song et al., A modified teaching-learning-based optimization algorithm for solving optimization problem, Knowledge-Based Systems (2020), doi: https://doi.org/10.1016/j.knosys.2020.106599

[29] Ya-jun Zhang, Ningjian Huang, Rober G. Radwin, Zheng Wang & Jingshan Li (2022) Flow time in a human-robot collaborative assembly process: Performance evaluation, system properties, and a case study, IISE Transactions, 54:3, 238-250, DOI: 10.1080/24725854.2021.1907489

[30] Ying Sun, Yuelin Gao. (2019). A Multi-Objective Particle Swarm Optimization Algorithm Based on Gaussian Mutation and an Improved Learning Strategy. doi:10.3390/math7020148

[31] Zeynel Abidin Çil, Zixiang Li, Suleyman Mete, Eren Özceylan, Mathematical model and bee algorithms for mixed-model assembly line balancing problem with physical human–robot collaboration, Applied Soft Computing, Volume 93, 2020, 106394, ISSN 1568-4946, https://doi.org/10.1016/j.asoc.2020.106394

[32] Zikai Zhang, QiuHua Tang, Rubén Ruiz, Liping Zhang, Ergonomic risk and cycle time minimization for the U-shaped worker assignment assembly line balancing problem: A multi-objective approach, Computers & Operations Research, Volume 118, 2020, 104905, ISSN 0305-0548, https://doi.org/10.1016/j.cor.2020.104905