Disassembly Line Balancing Using Recursive Optimization in Presence of Task-Failure

Rakshit Kumar Singh, Amit Raj Singh, Ravindra Kumar Yadav

Disassembly Line Balancing using Recursive Optimization in presence of Task-Failure

Rakshit Kumar Singh[1][0000-0002-1869-8772], A. R. Singh[2][0000-0003-2112-5948], Ravindra Kumar Yadav[3]

[1,2,3] Department of Mechanical Engineering, National Institute of Technology Raipur, G.E. Road, Raipur, Chhattisgarh, India – 492010
[1]rakshit3994@gmail.com
[2]amitrajsingh1@gmail.com

Abstract. Disassembly lines have to face task-failure situations due to the variability in quality of incoming product. Such failure violates the precedence relationship for the remaining task at downstream stations. Therefore, task failure requires corrective measure to improve the profitability of disassembly line. In this paper, a recursive optimization approach has been proposed to improve the profitability of disassembly lines, which takes corrective measure to determine optimal sequence of tasks. For this purpose, Teaching Learning Based Optimization (TLBO) algorithm has been used to find optimal sequences before and after task failure. To reduce the computational time required during recursion, the proposed solution approach is equipped with memoized list for finding corrective measure. A numerical illustration has been used to demonstrate the applicability of proposed solution approach which is capable to handle high variability in quality of incoming products.

Keywords: Disassembly, Task failure, probability-based stacking, Line balancing.

1 Introduction

Product recovery aims to retrieve parts and components from old or outdated products by disassembling products facing their end-of-life cycle into its constituent parts and components [1]. The disassembled parts face three end-of-life choices: recycling, remanufacturing and disposal. These end-of-life choices are made based on different objectives such as market demand for certain component, extraction of hazardous parts, retrieving parts and storing them in inventory for future use [2]. Thus, one decisive component of product recovery is disassembly which makes it crucial to perform disassembly in a way that meets various economic and environmental objectives for making product recovery sustainable. There are various settings of resources to perform disassembly such as single workstation (for high flexibility and low processing rate), disassembly cell (for moderate throughput and flexibility), and disassembly lines (for high disassembly rate and low flexibility).

Disassembly lines are very desirable in scenarios requiring high production rate, they also have certain associated benefits such as economies of scale, division of labor etc. [3]. The sequence in which various disassembly tasks are to be performed is critical to optimize the utilization of disassembly line resources, this problem is referred to as Disassembly Line Balancing Problem (DLBP) in the literature. The sequence of task assignment for line balancing can be obtained using exact solution approaches such as linear programming [4], Mixed Integer programming [5], second order cone programming [6]. DLBP belongs to NP-hard complexity of combinatorial optimization problem and, as the problem size increases it becomes time consuming to solve it using exact solution approaches [7]. Several authors have also made use of heuristics and meta-heuristics for solving DLBP under different scenarios such as ant colony optimization [8], simulated annealing [9], genetic algorithm [10], and particle fish swarm algorithm [11].

Although DLBP is similar to Assembly Line Balancing Problem (ALBP), DLBP has some additional complications such as operational considerations arising out of variations in incoming quality of the product [1]. Due to variability in quality of incoming product it might not be possible to perform some task on a given core (the product being disassembled). This situation is referred to as task-failure in DLBP literature and it can prevent the execution of other tasks on downstream workstations due to precedence relations of failed tasks and its successors. This variation in incoming quality of the product makes optimizing the utilization of disassembly line resources even more complicated. The first solution attempt in task-failure environment was provided by Gungor and Gupta [1]. They used weighted state network to find optimal disassembly sequence using Dijkstra's shortest path algorithm, the model was generated with the assumption of complete disassembly if possible, and the probability of task failures were assumed to be known and deterministic. The only other study in presence of task failure was done by [12], in their work authors attempted to improve the profitability of a disassembly line under task failure environment by introducing reactive rebalancing whenever a task failed. This reactive rebalancing was done on a sequence generated using predictive balancing, under predictive balancing solution approach a solution was found for given number of workstations to find optimal profit without considering task failure. Then the tasks in optimal sequence were failed one by one and a new optimal sequence was generated onward from the failed task under reactive line balancing scheme by relaxing cycle time constraint for the downstream workstations. Relaxing cycle time to generate reactive balance violates the necessary condition for paced line and introduces additional inefficiency in line resource utilization. Since the reactive balancing is done for only one optimal sequence obtained by predictive balance, it's a possibility that the solution obtained might be inferior to a solution having lower objective values in predictive balance and yet produce better overall results after task failure.

Considering the lacunae of the disassembly setting and solution strategy mentioned above, this article aims to provide a novel solution approach for optimizing the DLBP under task failure environment. For this purpose, Teaching Learning Based optimization (TLBO) meta-heuristic algorithm is utilized in recursive fashion to generate optimal disassembly sequence without violating the cycle time constraint. To reduce the

time for reactive balancing during recursion, memoization of optimal reactive sequence is incorporated thereby, reducing the overall execution time of the algorithm.

The remaining paper is organized as follows: In section 2 notations and problem definition for DLBP under task failure are provided. in section 3, the proposed algorithm utilizing TLBO for optimizing DLBP under task failure environment is described. Numerical illustration along with results of computational experiments are given in section 4, finally the conclusion and future scope are provided in section 5.

2 Problem description/formulation

The focus of this work is on determining disassembly sequence of tasks for a single product to maximize profit on a straight line, and complete disassembly is targeted. However, partial disassembly is allowed only after a task in a given disassembly sequence has failed. It is assumed that the incoming product supply is infinite, and parts released upon performing any given task in each product are known, the retrieved components are accepted in their current condition by demand source, only one product is disassembled on each parallel line, operators are multi-skilled. Other disassembly parameters such as task time, revenue generated are known and deterministic. The probabilities of task failure are known and deterministic. For failed tasks, task completion time and costs are fully incurred. Every station has a buffer space meant to store subassemblies (to facilitate stacking of work stations). Partial disassembly is allowed (by adding dummy task) only after a task fails in the disassembly sequence.

The notations for formulating DLBP under task failure and TLBO are as follows:

n	Total number of tasks of product.
a	Total number of possible subassemblies.
i	Task index.
prt_i	List of parts released by task i.
CT	Cycle time.
$cost_i$	Costs of performing task i.
rev_j	Revenue generated by releasing part j.
TS_k	Tasks that can be performed on subassembly k.
ST_i	List of subassemblies activated upon performing task i.
t_i	Time required to perform task i.
scc	Station cost coefficient
pf_i	Probability of failure of task i.
$`ca$	Set of currently activated subassembly for a solution string.
cdt	Updated set of candidate tasks based on precedence relationship and ca.
gen	Current iteration number of algorithm.
m	Population size for TLBO (common to main and failed optimization stages).
c_t	Teaching coefficient for teaching phase.
XC_l	Continuous decision vector**:** $\|XC\| = m, \|XC_l\| = n$.
XD_l	Discrete sequences of tasks generated using XC.
$XCF_{l,k}$	Continuous decision vector generated after sequentially failing tasks in

	XD:$\left\|XCF_{l,k}\right\| = n + 1(\because$ dummy task included), $\forall l \in (1,m), \forall k \in (1,\|XD_l\|)$
$XDFp_{l,k}$	List of completed tasks before a task failed (includes the failed task).
$XDFa_{l,k}$	Sequence of tasks performable corresponding to failed sequence $XDFp_{l,k}$
xdf^*	Optimal sequence of tasks corresponding to $XDFp_{l,k}$.
ML	Memoization list containing xdf^* found corresponding to every $XDFp_{l,k}$
$eval$	List containing evaluated values for every sequence in XD
$evalf$	List containing evaluated values for every sequence $XDFa_{l,k}$
P_{XDFa}	Probability of completing sequence $XDFa_{l,k}$
P_{cXDFa}	Conditional probability of completing sequence $XDFa_{l,k}$ given that one of the sequence in $XDFa_{l,k}$, $\forall k \in (1,\|XD_l\|)$ will happen.
WS_l	Total number of work stations after probability-based stacking ($\forall l \in (1,m)$).
$wsc_{l,w}$	List containing tasks assigned to station w in disassembly sequence XD_l
T_f	Task index of failed task.

To provide visual representation for precedence relationship, diagrams or graphs are used in DLBP. The two main types of diagrams used in the literature are part-based precedence (PPD) and task-based precedence diagram (TPD). The part-based precedence diagram provides parts' order based on their immediate predecessor [13] while TPD represents the order of tasks based on their immediate predecessors. Executing any given task in a TPD results in removal of one or more subassemblies or parts. By using either TPD or PPD multiple disassembly sequences can be generated. However, TPD representation is more clear and easier to use for generating solution strings, for this reason we used Transformed AND/OR graph (TAOG) based TPD representation developed by Koç *et al.*, [14]. In TAOG representation the subassemblies are represented by artificial nodes (A_i) and tasks are represented using normal nodes (B_i). These nodes are connected using two types of arcs, for any given node only one predecessor and one successor should be selected for execution. The arcs in TAOG represent two types of relationships, the arcs connected by a semicircle indicate an OR type relationship and the normal arcs indicate a normal type relationship.

3 Recursive optimization using TLBO

Most optimization algorithms have some algorithm-specific parameters, the improper tuning of these parameters may increase the computation time or yield a local optimal solution. Considering this fact, we make use of TLBO proposed by Rao et al. [15] as it does not need any algorithm-specific parameters apart from common control parameters like number of generations and population size. The pseudocode for the TLBO algorithm is provided below where the variable $'r'$ is a real number generated randomly in the range (0,1):

Algorithm 1. Procedure of TLBO algorithm

Input: $eval, m,$ max $\_gen, lb, lbc, ubc$
Initialize random population: $XC = [\,][for\ j\ \ in\ range\ (m)]$

i=0
while $\boldsymbol{t} < \boldsymbol{max\_gen}$:
  for $i = 1$ *to* m:
    choose XC_best based on eval
    determine XC_mean
    $XC_{i_{new}} = XC_i + r(XC_{best} - c_t XC\_mean)$
    Evaluate XC_new
    If $eval(XC\_new)$ is better than $eval(XC_i)$
      Replace XC_i with XC_new
    Randomly choose a solution XC_r
    If $eval(XC_i) < eval(XC_r)$
      $XC_{i_{new}} = XC_i + r(XC_i - XC_r)$
    else:
      $XC_{i_{new}} = XC_i - r(XC_i - XC_r)$
    Evaluate $XC_{i_{new}}$
    If $eval(XC_{i_{new}}) > eval(XC_i)$
      Replace XC_i with $XC_{i_{new}}$
  t+=1
output: $XC\_best, eval(XC\_best)$

The TLBO is inspired by the teaching-learning dynamics and is based on the influential effect of teacher on the learning output of a class. The algorithm involves two modes of evolution: first mode is through teacher and is known as teacher phase and second mode is through learners interacting with other learners and is therefore termed as learner phase. In TLBO the population consists of a group of learners and different design variables are analogous to different courses being offered to learners. The fitness value is analogous to the performance of learners in various courses. After initialization, best solution in the population is assigned the role of teacher. After determining the teacher, teaching and learning phases are executed for each solution string. At the end of each iteration, average performance and the teacher are updated.
TLBO for disassembly operates on real coded continuous decision vector XC_l of length equal to number of tasks in the precedence diagram. Based on sorted priority of continuous representation a discrete set of tasks XD_l is generated in accordance with precedence constraints. The tasks in XD_l are then assigned sequentially to workstations without violating cycle time constraint. If the cycle time constraint is violated, a new workstation is opened and the task is assigned to it. For n part problem the random numbers needed for generating a continuous representation for a partial sequence are $n + 1$. The position of these numbers is associated with the task index and the n+1 position is used for a dummy task, that can be assigned at any sequence with no resource requirement for a given product once a task in the given sequence fails. Once this dummy task for a product is assigned no other tasks can be performed on that product. The list of candidate tasks (cdt) is generated based on precedence constraint and the task with largest associated real number is assigned. This procedure is repeated until task $n + 1$ is assigned or until there are no more tasks in cdt.
Once XD for entire population is generated the sequences in XD are evaluated. Then, best performing solution string (XC_best) in XD is determined, for each member of the population teaching and learning phases are performed as stated in algorithm 1.

After completion of teaching and learning phase of a population member the new solution is evaluated and accepted or rejected based on greedy selection and the best solution is updated. After all members of population go through teaching and learning phase, next iteration is started. The evaluation of a disassembly sequence in XD require additional steps due to inclusion of task failure. The flow diagram of the proposed recursive algorithm incorporates these steps as shown in figure 1.

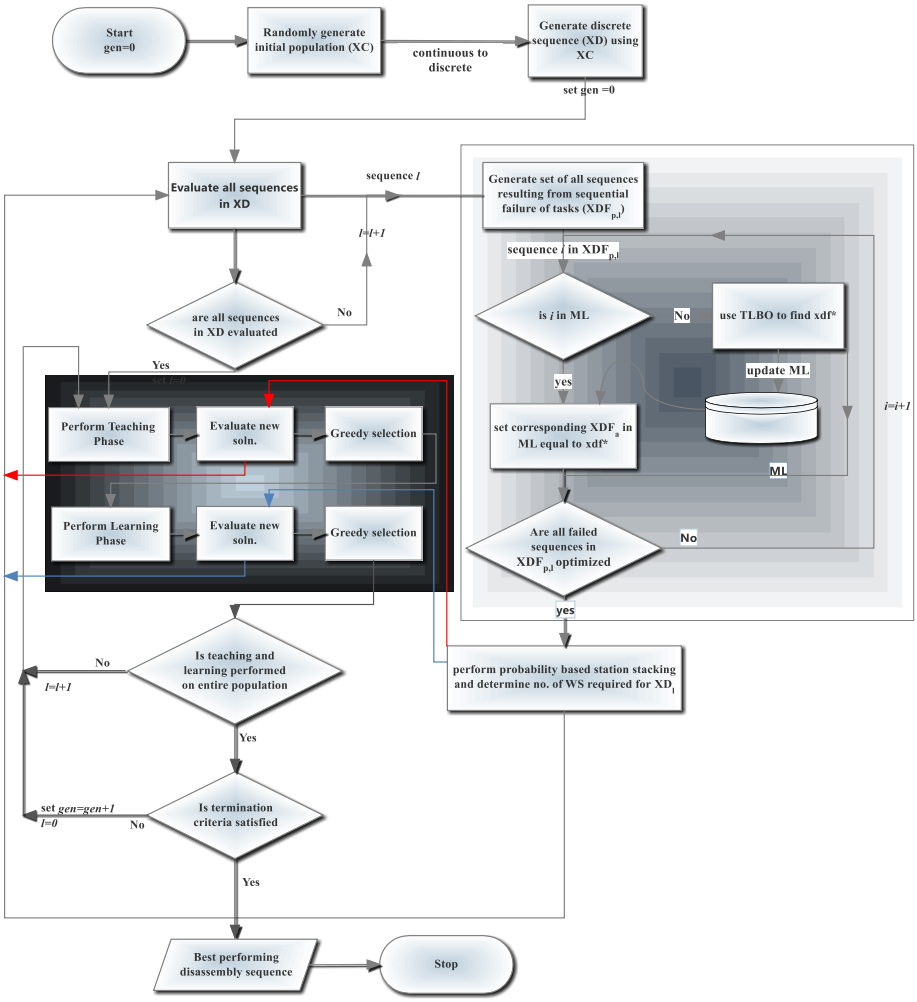

Fig. 1. Flowchart of recursive solution approach using memoization.

A sequence in XD is evaluated by performing the following steps,

1) The tasks of sequence XD_l are failed sequentially starting from the last task and then we start by searching optimal sequence in memoized list (ML) for each

failed sequence ($XDFp$). If $XDFp$ does not exist in ML then a new optimal sequence (xdf^*) is obtained using TLBO corresponding to $XDFp$ and the new found optimal solution corresponding to $XDFp$ is stored in ML.

2) For a given sequence in XD_l once all tasks are failed the resulting sequences ($XDFa$) are assigned to work stations based on task based-assignment strategy, the probability of occurrence of each sequence (eqn. 1) is calculated along with the associated profit of each sequence (eqn. 2).

$$P_{XDFa_{l,k}} = \prod_{i,i\neq T_f}\left(1 - pf_{XDFa_{l,k,i}}\right) * pf_{T_f}, \forall i \in XDFa_{l,k} \quad (1)$$

$$evalf = \sum_i rev_{i,i\neq T_f} - (scc * CT * ws + \sum_i cost_i), \forall i \in XDFa_{l,k} \quad (2)$$

3) Find the conditional probability of sequence k happening given that one of the sequences resulting from parent sequence l happens using equation 3. In other words, if a parent sequence l (sequence in which no task fails) is chosen for execution then the probability of having to perform a given resulting sequences (from task failure) is its conditional probability.

$$P_{cXDF_{l,k}} = \frac{p_{XDFa_{l,k}}}{\sum_k p_{XDFa_{l,k}}} \quad , k \in (1, |\mathrm{XD_l}|) \quad (3)$$

4) Determine the number of work stations required for a given sequence l by performing following steps:
   a) Determine probabilities of requiring different number of stations (using equation 4) based on number of workstations required for each sequence in $XDFa$ (corresponding to XD_l) and their associated conditional probabilities. This
   $$P_{ws_q,l} = \sum_{k,k\in(1,|XD_l|)} \begin{cases} P_{cXDFa_{l,k}} & , if\ ws_{cXDFa_{l,k}} \geq q \\ 0 & , else \end{cases}, \ \forall q \in (1, ws_{ub}) \quad (4)$$
   probability represents the chances of needing q or more stations for accommodating all failed sequences resulting from main sequence l.
   b) Set upper bound of number of work stations for any sequence in XD_l as the maximum number of workstations required amongst all corresponding failed sequences. For any sequence XD_l, the probabilities of requiring q number of workstations are calculated for corresponding failed sequences of XD_l.
   c) Perform stacking once the probabilities of requiring different number of stations is calculated. For this, if the sum of $P_{ws_{ub}}$ and $P_{ws_{ub-1}}$ is less than 1 then these stations are stacked and upper bound on station is set to upper bound minus one. If the stations are stacked then the probability of needing $q-1$ stations is updated as the sum of probabilities P_{ws_q} and $P_{ws_{q-1}}$.
   d) Perform stacking of workstations for a sequence XD_l iteratively until the probability of requiring last open work station exceeds 1.

5) Once stacking terminates, evaluate the fitness of XD_l as per equation 5.

$$eval_l = \sum_{k\in(1,|XD_l|)}\left(P_{XDFa_{l,k}} * \sum_{\forall i\in XDFa_{l,k}}(rev_{i,i\neq T_f} - cost_i)\right) - scc * WS_l \quad (5)$$

The first term in equation 5 represents the total profit without considering the cost of opening a workstation, the next term determines the station opening cost based on total number of stations required after stacking.

4 Numerical Illustration

The application of the proposed solution approach is demonstrated by using an example of an automatic pencil for balancing disassembly line in presence of task failure. The joint precedence diagram of the automatic pencil along with task times and task costs is provided in figure 2.

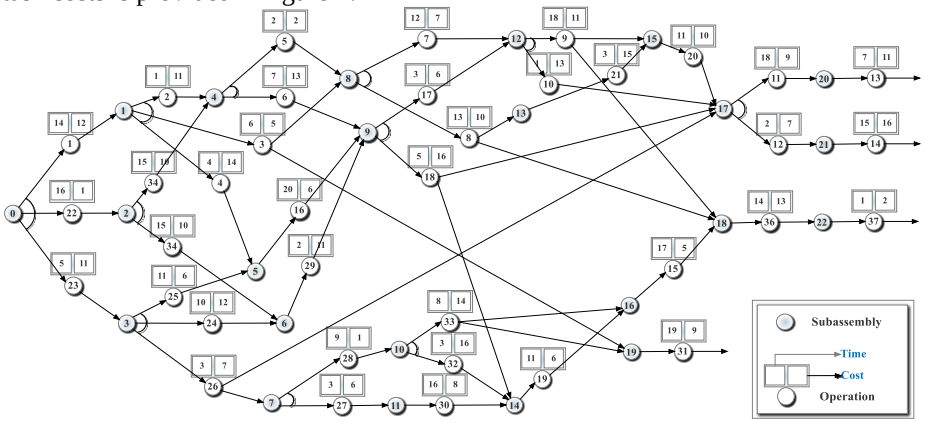

Fig. 2. precedence TAOG representation of automatic pencil example.

The recursive TLBO algorithm was coded in python and was run on Intel i5 4.2 GHz processor with 6-GB RAM. The input data for the problem instance such as revenues for released parts and probabilities of task failure are generated randomly. The remaining data for the problem instance is provided in Table 1. The task failure rates were generated in the range (0,0.1) and scc was assumed to be 7, the problem is solved for a cycle time of 30 seconds.

Table 1. Problem data for automatic pencil example

Part index *(j)*	1	2	3	4	5	6	7	8	9	10	11
Task releasing part *j*	22,28, 29,30, 34	5,17, 19,3	2,6,22, 24,27, 31	10,15, 20	36	37	37	11, 14	13, 14	12, 13	4,6,7,21, 23,35
Revenue	580	470	510	46	47	52	40	54	54	43	560

The optimal sequence and resulting optimal sequences due to failed tasks are given in Figure 3. The expected profit by the optimal sequence is 1379.61. The optimal sequence comprises of five workstations after stacking. The average execution time for finding the optimal disassembly sequence was recorded to be 198 seconds over 30 different runs for $m = 20$ and termination condition set as 100 iterations for TLBO.

	Failed task	WS 1	WS 2	WS 3	WS 4	WS 5	WS 6	Sequence probability	profit
Optimal XD	None	22	34,6,18	19,15	36,37	11,13	---	0.3329	3398
x_{df}^{*}	22	22,23,26,12,27	30,19	15	36,14,37	---	---	0.0495	2358
	34	22	34,35	29,18,19	15,12	36,37,14	---	0.0481	2919
	6	22	34,6,5	7,9	20,12,14	36,37	---	0.0071	2930
	18	22	34,6,18,17	9	36,37,20,12	14	---	0.0010	3427
	19	22	34,6,18,17	19	---	---	---	0.0832	2688
	15	22	34,6,18,17	19,15	---	---	---	0.0806	3153
	36	22	34,6,18,17	19,15	36	---	---	0.0339	3182
	37	22	34,6,18,17	19,15	36,37	---	---	0.0880	3227
	11	22	34,6,18,17	19,15	36,37	11,12	14	0.0381	3430
	13	22	34,6,18,17	19,15	36,37	11,13	---	0.0097	3359
Workstation stacking probabilities	Iteration 1	1	1	1	0.7861	0.5623	0.0497	Expected profit =1379.61	
	Iteration 2	1	1	1	0.7861	0.6120	---		
		Number of workstations needed after stacking=5							

Fig. 3. Optimal sequence and corresponding optimal sequences for failed tasks.

This execution time is significantly higher than the time consumed by reactive balancing strategy, but unlike reactive balancing the computation for finding optimal sequence is not performed every time a task fails in real time, instead all the calculations are performed only once and a solution sequence is generated for every scenario (in case of task-failure) without violating cycle time constraint.

5 Conclusion and future scope

In this paper, a recursive TLBO solution approach is developed to find a disassembly solution for optimizing expected profit, without violating the cycle time constraint as a reaction to task failure. To prevent the violation of cycle time constraint in event of a task failure, the number of workstations were determined for every sequence and then probability-based stacking of workstations was performed to determine number of workstations needed (which was required for expected profit evaluation). To reduce the time consumption of the recursive TLBO, memoization strategy was adopted and optimal sequences resulting from a failed task of parent sequence were stored and called whenever needed further down the iterative process. The proposed model was illustrated using an example of disassembly of automatic pencil. The result indicates that the proposed algorithm is able to find a near optimal solution within reasonable amount of time (average execution time of 3 min 18 seconds for thirty runs). As a future research direction, the proposed algorithm can be tested on a wide range of data set under different task failure rates and the effect of proposed solution strategy can be compared with that of a system in which predictive and reactive balancing are done separately. As another possible extension, the proposed solution approach can be tested under different line settings (such as parallel lines) to dampen the effect of uncertainty in quality of incoming product. Further, to reduce the problem complexi-

ty, only single task-failure was considered. Thus, for a more comprehensive analysis multiple task-failure setting can be studied.

References

1. A. Güngör and S. M. Gupta, “A solution approach to the disassembly line balancing problem in the presence of task failures,” *Int. J. Prod. Res.*, vol. 39, no. 7, pp. 1427–1467, 2001, doi: 10.1080/00207540110052157.
2. A. Gungor and S. M. Gupta, “Issues in environmentally conscious manufacturing and product recovery: A survey,” *Comput. Ind. Eng.*, vol. 36, no. 4, pp. 811–853, 1999, doi: 10.1016/S0360-8352(99)00167-9.
3. S. K. Das and R. Caudill, “The design of high volume disassembly lines,” in *Demanufacturing of Electronic Equipment for Reuse and Recycling Information Exchange Meeting Archives*, 1999, pp. 25–26.
4. A. J. D. Lambert, “Linear programming in disassembly/clustering sequence generation,” *Comput. Ind. Eng.*, vol. 36, no. 4, pp. 723–738, 1999, doi: 10.1016/S0360-8352(99)00162-X.
5. F. T. Altekin, Z. P. Bayındır, and V. Gümüşkaya, “Remedial actions for disassembly lines with stochastic task times,” *Comput. Ind. Eng.*, vol. 99, pp. 78–96, 2016, doi: 10.1016/j.cie.2016.06.027.
6. M. L. Bentaha, O. Battaïa, A. Dolgui, and S. J. Hu, “Second order conic approximation for disassembly line design with joint probabilistic constraints,” *Eur. J. Oper. Res.*, vol. 247, no. 3, pp. 957–967, 2015, doi: 10.1016/j.ejor.2015.06.019.
7. Gupta, S.M., & Lambert, A.J.D., “Disassembly Line Balancing,” *Environment Conscious Manufacturing (1st ed.)*. CRC Press, 2007, doi: 10.1201/9781420018790.
8. C. B. Kalayci and S. M. Gupta, “Ant colony optimization for sequence-dependent disassembly line balancing problem,” *J. Manuf. Technol. Manag.*, vol. 24, no. 3, pp. 413–427, 2013, doi: 10.1108/17410381311318909.
9. C. B. Kalayci and S. M. Gupta, “Simulated annealing algorithm for solving sequence-dependent disassembly line balancing problem”, vol. 46, no. 9. IFAC, 2013.
10. M. Seidi and S. Saghari, “The balancing of disassembly line of automobile engine using Genetic Algorithm (GA) in fuzzy environment,” *Ind. Eng. Manag. Syst.*, vol. 15, no. 4, pp. 364–373, 2016, doi: 10.7232/iems.2016.15.4.364.
11. Z. Zhang, K. Wang, L. Zhu, and Y. Wang, “A Pareto improved artificial fish swarm algorithm for solving a multi-objective fuzzy disassembly line balancing problem,” *Expert Syst. Appl.*, vol. 86, pp. 1339–1351, 2017, doi: 10.1016/j.eswa.2017.05.053.
12. F. T. Altekin and C. Akkan, “Task-failure-driven rebalancing of disassembly lines,” *Int. J. Prod. Res.*, vol. 50, no. 18, pp. 4955–4976, 2012, doi: 10.1080/00207543.2011.616915.
13. S. M. McGovern and S. M. Gupta, *Disassembly Line: Balancing and Modeling*. McGraw-Hill Education, 2011.
14. A. Koc, I. Sabuncuoglu, and E. Erel, “Two exact formulations for disassembly line balancing problems with task precedence diagram construction using an AND/OR graph,” *IIE Trans. (Institute Ind. Eng.*, vol. 41, no. 10, pp. 866–881, 2009, doi: 10.1080/07408170802510390.
15. R. V Rao, V. J. Savsani, and D. P. Vakharia, “Teaching–learning-based optimization: A novel method for constrained mechanical design optimization problems,” *Comput. Des.*, vol. 43, no. 3, pp. 303–315, 2011, doi: https://doi.org/10.1016/j.cad.2010.12.015.