Computers and Operations Research 124 (2020) 105064

Contents lists available at ScienceDirect

Computers and Operations Research

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

Balancing of two-sided disassembly lines: Problem definition, MILP model and genetic algorithm approach

Ibrahim Kucukkoc

Department of Industrial Engineering, Balikesir University, 10145 Balikesir, Turkey

ARTICLE INFO

Article history:
Received 1 January 2019
Revised 10 July 2020
Accepted 17 July 2020
Available online 26 July 2020

Keywords:
Disassembly line balancing
Two-sided lines
Mixed-integer linear programming
Genetic algorithm

ABSTRACT

The recovery of end of life (EOL) products has become an important issue in terms of economic as well as social and environmental considerations. Recent rigid environmental regulations also contribute to the popularity of disassembly and product recovery topics among academicians and practitioners. Disassembly lines have been utilised to break EOL products into pieces and remove parts which can be reused in the manufacturing of new products. However, to the best of the authors' knowledge, there is no research on the two-sided disassembly lines, which are used for disassembly of large-sized products. Therefore, this research contributes to literature by introducing the two-sided disassembly line balancing problem (TDLBP) and modelling it mathematically for the first time. The problem is depicted and the challenges are explored through extensive numerical examples. Secondly, a powerful genetic algorithm approach, called 2-GA, is developed for solving the introduced TDLBP considering complex AND/OR precedence relations. Computational tests are conducted to test the performance of the proposed 2-GA and the results are compared to those obtained from CPLEX and tabu search algorithm. From the comparison of the obtained solutions, it can be concluded that 2-GA has a superior performance in finding optimal (or at least near-optimal) solutions usually within less than one second.

1. Introduction

Recently, there has been an increasing interest in disassembly lines, on which end-of-life (EOL) products are partitioned into parts. This is majorly due to the shortened product life-cycles and attempts to decrease their effects on the environment and economy (Güngör and Gupta, 2002). Recycling EOL products and remanufacturing using valuable parts and/or components removed from EOL products help minimise total waste and maintain a sustainable production economy. For these purposes, dismantling the EOL products through a set of disassembly operations are the first and vital step.

A set of workstations is brought together via a material transportation system to construct a disassembly line on which disassembly operations are performed. Disassembly lines provide advantages such as high productivity and so increased utilisation of the resources (Kalayci and Gupta, 2013). The disassembly line balancing problem (DLBP) is the problem of partitioning tasks to workstations with the aim of optimising one or more performance criterion while satisfying the capacity and precedence constraints. Different from the assembly line balancing problems (ALBPs), DLBPs contain a much more complex precedence relationship constraint (as will be exemplified in Section 2). There is only AND precedence relationship in the ALBPs, whereas DLBPs have both AND and OR precedence relations, which makes the problem even harder to solve (which is already NP-hard). Some studies on DLBPs also consider OR successors (Koc et al., 2009). However, the current work follows the work of Güngör and Gupta (2002) and studies the DLBP without OR successor, which have a wide range of applications in real world (see, for example Ding et al. (2010), Kalayci and Gupta (2013) and Ren et al. (2017, 2018) among others). There also are many other differences between ALBPs and DLBPs as clearly explained by Lambert and Gupta (2005) and Hezer and Kara (2015). Therefore, as will be seen in the computational tests section of this work (Section 4), a more comprehensive as well as efficient solution methodology is needed for solving the DLBPs (Özceylan, 2018).

The two-sided disassembly lines are constructed for dismantling large-sized products, such as large electrical households or vehicles used for transportation. Fig. 1 provides a schematic representation of a two-sided disassembly line. As seen from the figure, the workstations are located across both sides of the line (i.e. left and right). The EOL product enters the line from the first workstation and parts/components are removed sequentially by the operators located in the workstations. Each workstation needs to

E-mail address: ikucukkoc@balikesir.edu.tr

https://doi.org/10.1016/j.cor.2020.105064
0305-0548/© 2020 Elsevier Ltd. All rights reserved.

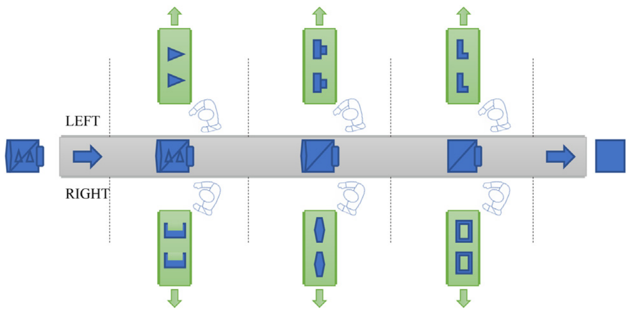


Fig. 1. The configuration of a two-sided disassembly line.

accomplish assigned tasks into it within the maximum time allowed. As the tasks are handled on both sides, the complex precedence relationships must be handled considering the predecessors of a task, which may be located on the opposite side of the line. The details on the complex relationships and more on the problem definition will be presented in Section 2.

DLBP was introduced by Güngör and Gupta (1999) and a shortest-path formulation was proposed by Gungor and Gupta (2001) to solve the DLBP taking the task failures into account. The DLBP has attracted many researchers since then. McGovern and Gupta (2007) have shown that the DLBP is NP-hard in the problem complexity. Exact methods, mainly utilising mixed-integer linear programming (MILP) models, have been developed by researches (Ren et al., 2018). Bentaha et al. (2015) proposed an exact solution approach for solving the DLBPs under uncertainty. In their research, task times have been assumed to be random variables with known normal probability distributions. Also, Mete et al. (2016) developed a mathematical model for solving the DLBP considering resource dependent constraints. Kucukkoc et al. (2020) introduced Type-E multi-manned DLBP and proposed efficient linear and non-linear models to solve the problem. The MILP model proposed by Altekin et al. (2008) aimed at maximising the profit while determining the parts whose demand is to be fulfilled to generate revenue. Two exact formulations that utilise an AND/OR graph as the main input for the purpose of maintaining the feasibility was proposed by Koc et al. (2009). A two-stage approach was also developed by Altekin and Akkan (2012) to rebalance the disassembly lines in the case of task-failure. Paksoy et al. (2013) took into account the fuzzy goals when balancing DLBPs. A mathematical model and an ant colony approach were developed by Mete et al. (2018) to balance a hybrid line consisting of both assembly and disassembly tasks. Li et al. (2019) developed a fast branch, bound and remember algorithm to obtain state-of-the-art results for simple DLBPs.

Heuristics and metaheuristics have also been developed to get timely-manner solutions for large-size problems. To cite a few, beam search algorithm based solution approach was proposed by Mete et al. (2016) for the DLBP. McGovern and Gupta (2003) presented a greedy/2-opt hybrid algorithm to solve the multi-objective DLBP, and Ren et al. (2018) extended this method to solve the multi-objective DLBP with weights-based multi-criteria decision.

The metaheuristics were mainly on the multi-objective DLBPs. McGovern and Gupta (2007) developed a genetic algorithm based approach for the multi-objective DLBP. Kalayci et al. (2014) used tabu search for the sequence-dependent DLBP and Kalayci et al. (2016) developed a hybrid neighbourhood search method to deal with the sequence-dependent task times in DLBP. A variable neighbourhood search method was developed by Ren et al. (2018) for the multi-objective DLBP. Swarm-based algorithms have also been developed for the variants of the DLBP. See, for example; Agrawal and Tiwari (2008), Ding et al. (2010), Kalayci and Gupta (2013) and McGovern and Gupta (2006) for ant colony optimisation; Kalayci and Gupta (2013), Kalayci et al. (2015) and Liu and Wang (2017) for artificial bee colony algorithm; Kalayci and Gupta (2013) and Xiao et al. (2017) for particle swarm optimisation algorithm; Zhang et al. (2017) for artificial fish swarm algorithm; Zhu et al. (2018) for pareto firefly algorithm; and Liu et al. (2018) and Ren et al. (2017) for other evolutionary algorithms. A recent survey by Özceylan et al. (2018) provides a comprehensive review of the algorithms applied to DLBPs.

Interestingly, this review of the literature makes it clear that no research was conducted on the balancing of two-sided disassembly lines. While two-sided ALBPs have been studied extensively (e.g. see Bartholdi (1993), Li et al. (2017), Kucukkoc and Zhang (2015a), Make et al. (2016), Li et al. (2020)), to the best of the authors' knowledge, the two-sided disassembly line balancing problem (called TDLBP hereafter) has never been studied in the literature. Thus, the basic motivation of this paper is the lack of research on TDLBP. This paper mainly contributes to literature by introducing the TDLBP and mathematically modelling it for the first time in the literature. Thus, a new disassembly line balancing problem type is brought to the interest of both academia and industry. A MILP model, which is capable of solving the TDLBP optimally for the small and some medium-sized problems, is developed to formally describe the problem. Moreover, a new genetic algorithm approach, called 2-GA, is developed considering the comprehensive AND/OR precedence relations on a complex line configuration. It is particularly worthy to note that a special scheduling mechanism is needed in the solution methodology since tasks performed on opposite sides of the two-sided line may have AND as well as OR precedence relations in between. The solution building mechanism of 2-GA is clearly described by pseudo-codes and illustrated through numerical examples. A set of 88 instances is generated considering the specifications of TDLBP and a comprehensive computational study is carried out to test the performance of the proposed 2-GA. The results obtained by 2-GA are compared to those obtained from CPLEX and tabu search (TS). The results demonstrate the superiority of the 2-GA in both solution building capacity and search speed.

The rest of the paper is organised as follows. Section 2, describes the TDLBP and formulates it via a MILP model. The challenges are explored in detail and the optimal solution of a numerical example is also provided. Section 3 explains the mechanisms of the proposed 2-GA methodology in details via a numerical example. The results of the computational study conducted to test the performance of 2-GA are presented in Section 4 and the paper is concluded in Section 5 together with some future research directions and managerial implications. Data on test problems and the pseudocode of TS are provided in Appendices.

2. Problem statement and mathematical model

This section introduces the TDLBP and proposes a MILP model to formulate it with the aim of minimising a weighted summation of more than one performance criterion.

2.1. Problem description

The TDLBP is composed of balancing a two-sided disassembly line across which a series of workstations are utilised on both left and right sides (as presented in Fig. 1 above). Each pair of workstations on left and right are called a mated-station. The EOL product enters the line from the first mated-workstation and parts are sequentially removed by operators located in the workstations.

The products are of large-size so they require a two-sided line to prevent operators from wasting their time by repeatedly walking around it.

Each removal task usually has a preferred operation direction (or side) to be performed, i.e. 'Left (L)' or 'Right (R)'. Some tasks may be located on either left or right side, called 'Either (E)' side task. Each task ($i \in I$) requires a certain amount of processing time, t_i, to remove a part. Each workstation processes at least one removal task within the time allowed, called cycle time (CT). Thus, the total workload of each workstation cannot exceed the cycle time. Also, the tasks must be completed within the cycle time, considering the unique starting time of each task. This is particularly important because tasks assigned to a specific side of the line may be preceded by those assigned to the other side of the line.

There are two kinds of precedence relations between tasks: (i) AND precedence and (ii) OR precedence. AND precedence relation requires that all predecessors of a specific task must be assigned and completed prior to the initialisation of that task. See the precedence relationship diagram of 2P22-OR problem presented in Fig. 2 (adapted from a well-known problem, P22-OR (Kalaycılar et al., 2016)). The arrows (with no arc) represent the AND precedence relation. For example, there is an AND precedence relation between tasks 3 and 4. Hence, task 3 must be completed before the initialisation of task 4.

From the view of the OR precedence relation, it is satisfactory to assign and complete at least one of its OR predecessors to initialise a specific task. The arrows with an arc in the figure denote the OR precedence relations. For example, there is an OR precedence relation between task 22 and its predecessors (tasks 2 and 4). Therefore, to be able to start task 22, it is satisfactory to complete either task 2 or task 4. Thus, the earliest starting time of a task can be generalised as follows. Assume that ANDP(i) is the set of AND predecessors of task i. Similarly, ORP(i) is the set of OR predecessors of task i. The calculation for the earliest starting time of task i can be generalised as $EST(i) = \max\left\{ \min_{g \in ORP(i)} \left\{ t^c_g \right\}, \max_{h \in AND(i)} \left\{ t^c_h \right\} \right\}$, where t^c_g denotes the completion time of task g, which is equivalent to the sum of its earliest starting time and processing time, i.e. $t^c_g = EST(g) + t_g$. This condition differentiates the TDLBP from the TALBP and makes the problem, which is already NP-hard, even harder to solve. Fig. 3 presents this situation through a possible task assignment with and without using the OR precedence. As seen from the figure, task 22 can start as early as $t^s_{22} = 38$ time units (after the completion of one of its OR predecessors, i.e. task 2) in a TDLBP with OR precedence relation. However, its earliest start time would be $t^s_{22} = 55$ if the problem was a TALBP (without an OR precedence relationship between task 22 and its predecessors). As seen clearly, the existence of the OR relations has the potential of increasing the number of possible combinations of task assignments to workstations, which would eventually help shorten the line and minimise the number of workstations.

A MILP model is developed for balancing the TDLBP carefully handling the abovementioned constraints with the aim of minimising the number of mated-stations (or line length, in other words) as a primary objective and the number of workstations as a secondary objective. The following assumptions are made to develop the model. First of all, the tasks and their processing times are known and deterministic. Only one operator may be utilised in each workstation (no parallelisation is allowed). Neither work in process is allowed between workstations nor breakdowns/failures are considered in the model. The cycle time is assumed to be known and deterministic and walking times are ignored.

2.2. Mathematical model

This section presents the MILP model developed for solving the TDLBP optimally. The notation is presented as follows.

Indices:	
i, h, p	Task index.
j, g	Mated-station index.
k, f	Side of the line; $k, f = \begin{cases} 1 & \text{if the side is left} \\ 2 & \text{if the side is right} \end{cases}$.
(j, k)	The k side workstation of the mated-station j.
Parameters:	
I	Set of tasks in the precedence diagram, $I = \{1, 2, \cdots, i, \cdots, nt\}$.
J	Set of mated-stations, $J = \{1, 2, \cdots, j, \cdots, nm\}$.
CT	Cycle time
AL	Set of tasks which should be performed at a left-side workstation, $AL \subseteq I$.
AR	Set of tasks which should be performed at a right-side workstation, $AR \subseteq I$.
AE	Set of tasks which can be performed on left or right side of a mated-station, $AE \subseteq I$.
ANDP(i)	Set of AND predecessors of task i.
ANDPT	Set of tasks which have AND predecessors.
ORP(i)	Set of OR predecessors of task i.
ORPT	Set of tasks which have OR predecessors.
$K(i)$	The set of integers which indicate the preferred operation direction of task i, $K(i) = \begin{cases} \{1\} & \text{if } i \in AR \\ \{2\} & \text{if } i \in AL \\ \{1,2\} & \text{if } i \in AE \end{cases}$.
ψ	A large positive number.
t_i	Processing time of task i.
Decision variables:	
t^s_i	Start time of task i.
x_{ij}	1, if task i is assigned to mated-station j; 0, otherwise.
w_{ik}	1, if task i is assigned to side k; 0, otherwise.
z_{hi}	1, if task h precedes task$i$ in the precedence diagram or task h is operated earlier than task i when both tasks are allocated to the same workstation; 0, otherwise.
F_j	1, if both sides of mated-station j are utilised; 0, otherwise.
G_j	1, if only one side of mated-station j is utilised; 0, otherwise.
U_{jk}	1, if workstation (j, k) is utilised; 0, otherwise.

The model is presented as follows on the basis of the relative order time. Namely, the tasks in the first mated-station is finished within $[0, CT]$, and the tasks in the second mated-station is operated within $[CT, 2CT]$.

$$\text{Min } w_1 \sum_{j \in J} (F_j + G_j) + w_2 \sum_{j \in J} \sum_{k=1,2} U_{jk} \tag{1}$$

$$\sum_{j \in J} x_{ij} = 1 \; \forall i \in I \tag{2}$$

$$\sum_{k \in K(i)} w_{ik} = 1 \; \forall i \in I \tag{3}$$

$$t^s_i + \psi(1 - z_{hi}) \geq t^s_h + t_h \; \forall i \in I, \; h \in I \wedge i \neq h \tag{4}$$

$$z_{hi} + z_{ih} = 1 \; \forall i \in \text{ANDPT}, \; h \in \text{ANDP}(i) \tag{5}$$

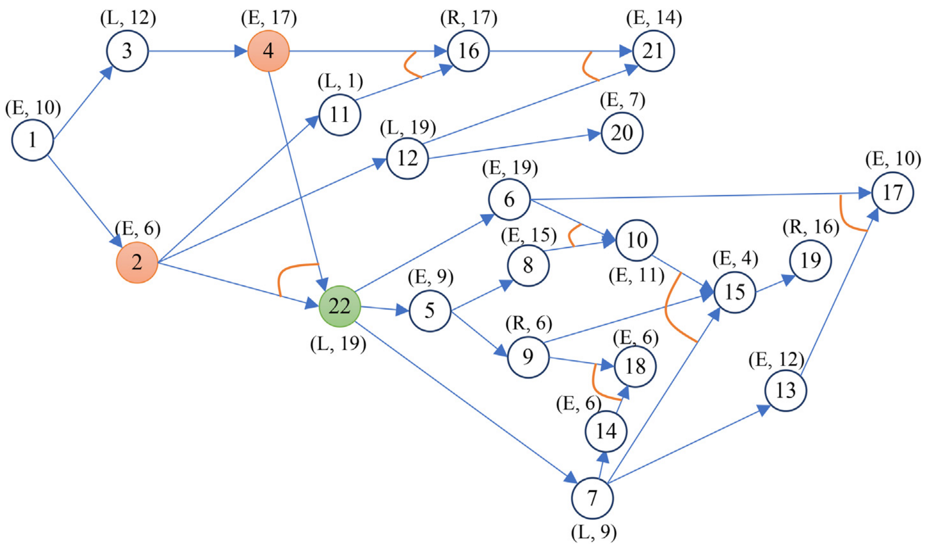


Fig. 2. Representation of the precedence relations for 2P22-OR problem.

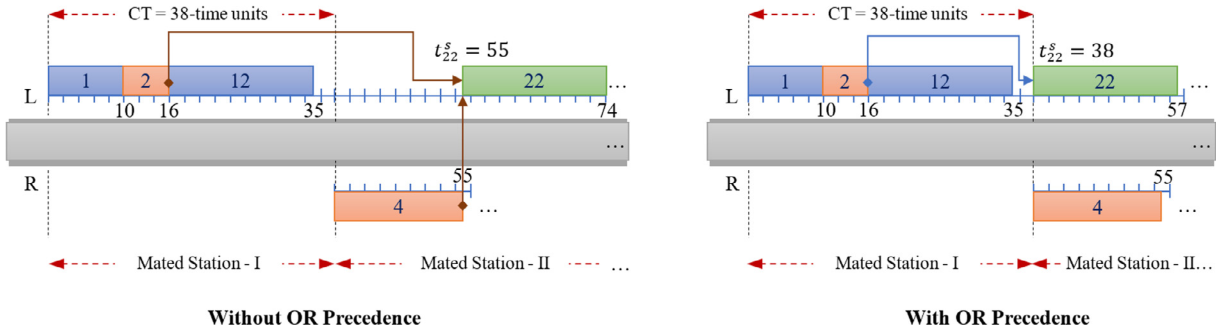


Fig. 3. The representation of having an OR relation on a two-sided line.

$$z_{hi} = 1 \; \forall i \in \text{ANDPT}, \; h \in \text{ANDP}(i) \tag{6}$$

$$\sum_{h \in ORP(i)} z_{hi} \geq 1 \; \forall i \in \text{ORPT} \tag{7}$$

$$z_{ih} + z_{hp} - 1 \leq z_{ip} \; \forall i, h, p, i{\neq}h, \; h{\neq}p \text{ and } i{\neq}p \tag{8}$$

$$z_{ih} + z_{hi} \leqslant 1 \; \forall i \in I, \; h \in I \wedge i{\neq}h \tag{9}$$

$$z_{ih} + z_{hi} + \psi(2 - x_{ij} - x_{hj}) + \psi(2 - w_{ik} - w_{hk}) \geq 1$$

$$\forall i \in I, \; h \in I \wedge i{\neq}h, \; j \in J, \; k \in K(i) \cap K(p) \tag{10}$$

$$t^s_p + \psi(2 - x_{ij} - x_{pj}) + \psi(2 - w_{ik} - w_{pk}) + \psi(1 - z_{ip}) \geq t^s_i + t_i$$

$$\forall i \in I, \; p \in I \wedge i{\neq}p, \; j \in J, \; k \in K(i) \cap K(p) \tag{11}$$

$$t^s_i \geq CT\left(\sum_{j \in J} ((j-1)x_{ij})\right) \forall i \in I \tag{12}$$

$$t^s_i + t_i \leq CT\left(\sum_{j \in J} (j \, x_{ij})\right) \forall i \in I \tag{13}$$

$$U_{jk} \geq x_{ij} + w_{ik} - 1 \; \forall i \in I, \; j \in J, \; k = 1, 2 \tag{14}$$

$$\sum_{k=1,2} U_{jk} - 2F_j - G_j = 0 \; \forall j \in J \tag{15}$$

$$(F_j + G_j) \geq (F_{j+1} + G_{j+1}) \; \forall j \in J \wedge j < nm \tag{16}$$

Equation (1) minimises the weighted mated-station number and workstation number, where w_1 and w_2 are two parameters set to 100 and 1, respectively. Thus, the length of the line is minimised ensuring that the mated-stations are loaded prior to opening a new one. Constraint (2) and constraint (3) indicate that a task must be allocated to exactly one mated-station and one side. Constraints (4–8) deal with the precedence constraints, indicating that task i can be started only when its AND predecessors have been completed (in Equation (6)) and at least one of its OR predecessors has been completed (in Equation (7)). Constraint (8) donates that task h is executed before task p when task h is executed before task i and task i is executed before task p. Constraints (9–11) restrain the task assignment on a workstation, and Equation (11) is reduced to $t^s_p \geq t^s_i + t_i$ when task i is allocated before task p and they are allocated to the same workstation. Constraint (12) and constraint (13) deal with cycle time constraint, indicating that a task is operated during the available start and end time of the corresponding mated-station. Constraint (14) ensures that $U_{jk} = 1$ when there is at least one task allocated to workstation (j, k). Constraint (15) calculates F_j and G_j, and constraint (16) indicates that a mated-station can be utilised only when its former mated-station is utilised. Note that the constraints pertaining to precedence relationship are different from those in TALBPs.

2.3. Numerical example

The 2P22-OR problem, for which the precedence relationship diagram was given in Fig. 2 (see Section 2.1), was attempted to be solved using the MILP model developed. For this aim, the model was coded in General Algebraic Modelling Software (GAMS) v23.0 and run on a PC equipped with Intel® Core™ i7-6700HQ CPU @2.60 GHz and 16 GB of RAM. However, the CPLEX solver did not produce any solution within 11,000 s, which is a considerably large period of time. Therefore, a 10-task problem, 2P10-OR (modified from the well-known P10-OR (McGovern and Gupta, 2003)) was solved using the CPLEX solver embedded in GAMS. The problem data and the optimal solution of the problem are presented in Fig. 4 (a dummy part –A1– with no processing time is given to simplify the presentation of the precedence relations). The cycle time was considered to be $CT = 36$-time units and the optimum solution was obtained within 1 s.

As seen from the solution, all constraints have been satisfied. A total of five workstations have been utilised in four mated-stations and the objective value is found to be 405 (based on the values of parameters w_1 and w_2). The right-side workstations in Mated Stations I and III and the left side workstation in Mated Station II are fully loaded (36-time units). On the other hand, the workstations on the right side of Mated-stations II and IV and the left side of the Mated-station I have not been utilised, no tasks have been assigned in other words. There exists a total of seven-time units idle time (four- and three-time units on the left of the Mated Stations III and IV, respectively). The efficiency of the line can simply be calculated as $LE(\%) = 100 \times \sum_{i \in I} t_i / (K \times C) = 100 \times 173/(5 \times 36) \cong 96.11\%$, which is considerably high.

3. Genetic algorithm based approach

This section explains the GA based approach, called 2-GA, proposed for solving the TDLBP especially when the mathematical models fall short. In the following subsections, the outline of the 2-GA is provided first, followed by the details of the new solution building and search mechanisms.

3.1. Outline of the 2-GA

GA is an evolutionary algorithm inspired by the survival of the fittest. Its stochastic search mechanism enables the emergence of new individuals derived mostly from the fittest individuals, called parents. So that the natural selection is imitated as the fittest individuals are selected for reproduction in order to form the next generation. GA is selected in this research due to its high performance in solving the combinatorial engineering problems. In addition to its successful implementations on DLBPs (as given in Section 1), it has been successfully applied for solving various complex design and optimisation problems from transportation (Gen et al., 2006; Vignaux and Michalewicz, 1991) to scheduling (Sadegheih, 2006) and ALBPs (Kucukkoc and Zhang, 2015b,c).

The outline of the 2-GA is presented in Fig. 5 as a pseudo-code. Details on sub methods will be provided after the explanations on the general outline of the algorithm.

As seen from Fig. 5, the algorithm is initialised and the initial population is generated building up a total of *popSize* chromosomes (note that the procedure followed when building a feasible chromosome will be presented in Section 3.2). Duplication of chromosomes is not allowed to have a diverse population. Each chromosome is then decoded using a novel approach considering the unique feature of the TDLBP to evaluate and calculate fitness values (the procedure will be given in Section 3.3). This process plays a vital role in the success of the implemented algorithm as the problem contains complex AND/OR precedence relations and the tasks are performed on both sides of the line, which requires a sophisticated scheduling effort.

Tournament selection is applied to select parents from the population to undergo genetic operators (crossover and mutation). The tournament size is determined by $[0.15 \times \text{popSize}]^+$ where $[X]^+$ denotes the smallest integer greater than or equal to X. The value of this parameter is determined with the consideration of preserving diversity while giving higher chance to the fittest individuals for reproduction. More details on genetic operators will be provided in Section 3.4.

Following the application of the genetic operators, a number of new individuals will be obtained. Prior to evaluating their fitness values, each new individual is checked and it is deleted if it duplicates any chromosome in the population. To form the new generation, the worst chromosomes in the population are replaced by the best children comparing their fitness values, as long as there is any newly obtained better individual (will be explained in detail

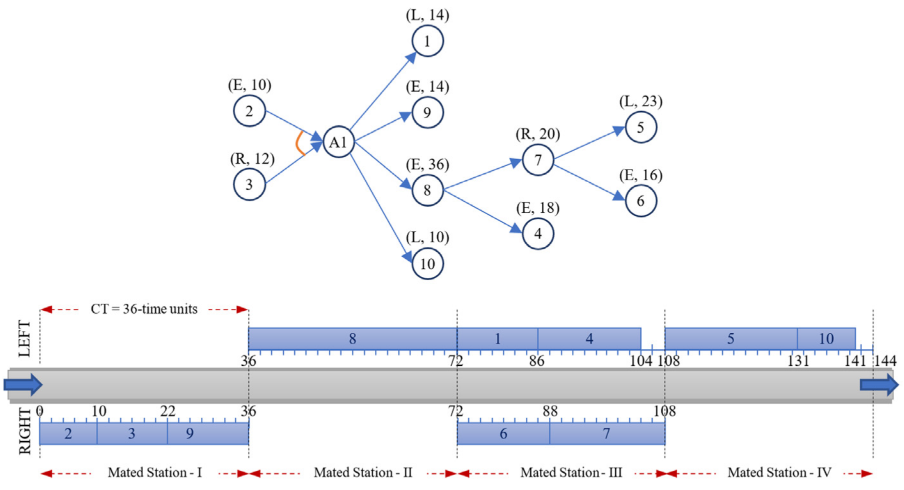


Fig. 4. The problem data for the 2P10-OR problem and its optimal solution.

```
Algorithm  2-GA()
1          Initialise
2          Generate Initial Population
3          Evaluate Chromosomes (Calculate Fitness)
4          iter := 1
5          WHILE (iter ≤ maxIter)
  5.1        Select Parents using Tournament Selection
  5.2        Apply Genetic Operators (Permutation Crossover, Insert&Swap
              Mutation)
  5.3        Repair Mutants after Mutation
  5.4        Remove Duplications
  5.5        Evaluate New Individuals (Calculate Fitness)
  5.6        Form the New Generation (Replace the Worst in the Population)
  5.7        iter++
6          ENDWHILE
7          Report the Best in the Population
```

Fig. 5. The general outline of the proposed algorithm.

in Section 3.5). This cycle continues until the maximum iteration number is exceeded and the best solution is reported.

3.2. Encoding

The procedure used for generating a feasible sequence, chromosome, is presented in Fig. 6. The chromosomes are built considering the precedence relationships to prevent infeasibility. For this aim, an array called `Pred[]` is updated after assigning a new task to determine which tasks are available in terms of the precedence relations. The term `nbTasks` represents the number of tasks to be assigned. When a task is selected and assigned to the chromosome, denoted with `chrom{}`, the temporary precedence relationship matrix, `TempPRMatrix[][]`, is updated and available tasks are determined based on the updated `Pred[]` array. A task can be available if it has no unassigned AND predecessor or at least one of its OR predecessors (if any) has been assigned. A submethod, `selectTask()`, is invoked to determine available tasks based on the `Pred[]` array and select a task. The pseudo-code of this submethod is exhibited in Fig. 7. When a task is selected, it is marked as assigned with the help of `TempTasks[]` array, and the status of its both AND and OR successors have been updated on the temporary `TempPRMatrix[][]`. The returned value, i.e. `selectedTask` by `selectTask()` method, is added to the chromosome and this procedure is continued until all tasks have been sequenced.

Fig. 8 presents a feasible chromosome sample for 2P22-OR problem, whose problem data was given in Fig. 2. As seen from the chromosome, the sequence does not violate the precedence relations. To give an example, tasks 2 and 4 are the OR predecessors of task 22 (as given in Fig. 2). The assignment of task 2 prior to task 22 is enough to sustain feasibility in terms of the precedence relations of this particular task.

```
Submethod       buildChrom()
1               Initialise
2               chrom := {}
3               WHILE (chrom.size() < nbTasks)
  3.1             i := 1
  3.2             WHILE (i ≤ nbTasks)  //Determine Pred[] array for precedences
    3.2.1           total1 := 0, total2 := 0, total3 := 0; j := 1
    3.2.2           WHILE (j ≤ nbTasks)
      3.2.2.1         IF (TempPRMatrix[j][i] = 1)
                        total1++
      3.2.2.2         ELSE IF (TempPRMatrix[j][i] = 2)
                        total2++
      3.2.2.3         ELSE IF (TempPRMatrix[j][i] = 3)
                        total3++
      3.2.2.4         ENDIF
      3.2.2.5         j++
      3.2.2.6       ENDWHILE
    3.2.3           IF (total1 > 0)
      3.2.3.1         Pred[i] := total1
    3.2.4           ELSE IF (total3 > 0)
      3.2.4.1         Pred[i] := 0
    3.2.5           ELSE IF (total2 > 0)
      3.2.5.1         Pred[i] := total2
    3.2.6           ELSE
      3.2.6.1         Pred[i] := 0
    3.2.7           ENDIF
    3.2.8           selectedTask := selectTask() //call submethod to select a task
    3.2.9           Add selectedTask to chrom{}
    3.2.10          i++
  3.3             ENDWHILE
4               ENDWHILE
```

Fig. 6. The procedure used for building up a feasible chromosome.

```
Submethod        selectTask()
1              Initialise, reset TempTasks[] and TempPRMatrix[][]
2              selectedTask := -1, avTasks := {}
3              count: = 0
4              WHILE (count < nbTasks) //check all available tasks
  4.1            IF (Pred[i] = 0 and TempTasks[i] != 0)
  4.2              Add TempTasks[i] to avTasks{}
  4.3            ENDIF
  4.4            Randomly select a task from avTasks{} and assign to selectedTask
  4.5            i := 1
  4.6            WHILE (i < nbTasks) //update precedence issues
    4.6.1          IF (TempPRMatrix[selectedTask][i] = 2
      4.6.1.1        TempPRMatrix[selectedTask][i] := 3
    4.6.2          ELSE
      4.6.2.1        TempPRMatrix[selectedTask][i] := 0
    4.6.3          ENDIF
    4.6.4          i++
  4.7            ENDWHILE
  4.8            TempTasks[selectedTask] := 0
  4.9            count++
5              ENDWHILE
6              return selectedTask
```

Fig. 7. The procedure used for selecting a task.

1	2	12	20	3	21	11	22	7	14	4	16	15	19	6	18	5	17	13	8	10	9

Fig. 8. Representation of a feasible chromosome sample.

3.3. Decoding

The chromosomes created to generate the initial population and those obtained after the genetic operators are evaluated using the decoding procedure given in Fig. 9. The decoding procedure applies a stochastic task allocation mechanism to explore the search space more effectively. The tasks that can be assigned to either side are assigned to left or right side, by a random decision given by the computer. The tasks on the chromosome are assigned to workstations based on their sequence on the chromosome.

The assignment procedure starts from the first task on the chromosome. It is assigned to its preferred side (left or right), if any. If it is an E type task, the assignment process starts from a randomly selected side. When a task is assigned, the current time holder of the current position (`ctLeft` or `ctRight`) is incremented by '$es + tasktime$'. The term `es` corresponds to the earliest starting time of the task, calculated by $es = max(ctLeft, max(ESTOR[task], ESTAND[task]))$. So, the earliest starting time is the maximum of `ctLeft`, `ESTOR[]` and `ESTAND[]` value. `ESTOR[]` and `ESTAND[]` hold the completion time of the task's OR and AND predecessors. By this way, the tasks that may have been assigned to the other side of the line are also taken into account in terms of the precedence relations. A task can only be assigned to the current side if $es + TaskTimes[task] \le CT$. The task is marked as assigned and the corresponding current time holder (`ctLeft` or `ctRight`) is updated. It is very important to update the `ESTOR[]` and `ESTAND[]` arrays every time a new task is assigned. To update `ESTAND[]` array, the array values of all AND successors of the assigned task are set to the completion time of the task. As for the `ESTOR[]` array, the array values of the OR successors are set to the minimum completion time among the corresponding task's OR predecessors.

If a task in the sequence cannot be assigned to the current mated station due to insufficient capacity, a new mated-station is opened (`nbMS++`) and the values of `ESTOR[]` and `ESTAND[]` arrays, and the current time holders `ctLeft` and `ctRight` are reset to zero. The task is assigned to the preferred side (if no preference, selected randomly) and this cycle continues until all tasks are assigned.

Fig. 10 presents the decoded assignment solution of the best chromosome obtained by 2-GA after 5000 iterations (CT = 37-time units). As seen from the solution, seven workstations are utilised within four mated-stations and the idle times are very small.

3.4. Genetic operators and repairing

Crossover is applied considering the permutation encoding nature of the chromosomes. Two parents are selected from the population via tournament selection and one-point permutation crossover is applied as illustrated in Fig. 11. A random cutting point is determined and the parents are divided into two parts, i.e. head and tail. *Head1* and *Head2* construct the heading parts of *Child1* and *Child2*, respectively. The tails of the children are completed deriving the missing genes as in the sequence they appear in the other parent. For example, the missing genes for *Child1* are added in the tail of *Child1* as in the sequence that they appear on *Parent2* to obtain a complete as well as a feasible chromosome. Thus, the chromosome does not violate the precedence relations. The missing genes for *Child2* are also completed similarly (based on their sequence on *Parent1*) and two complete individuals (children) are obtained after each crossover. This process is repeated by $CR \times popSize$ times (where *CR* represents the crossover rate) and children are kept (after removing duplicated ones, if any) for fitness evaluation.

Insert and swap mutations are applied to parents selected from the population. A random number, *rnd*, between [0–1) is determined and insert mutation is applied if $0 \le rnd < 0.5$. On the contrary, swap mutation is applied if $0.5 \le rnd < 1$. To apply the insert mutation, two random numbers (i.e. $r_1 < r_2$) are determined between 1 and the length of the chromosome. The gene at location r_1 is removed and relocated at location r_2. As for the swap mutation, the genes at randomly determined locations (i.e. r_1 and r_2) are swapped. Both mutation strategies are presented in Fig. 11. Note that the newly obtained children after mutation (mutants) may be infeasible in terms of the precedence relations. Therefore, a problem specific repair procedure is applied following the mutation. After the repairing process, it is ensured that every task is sequenced after all of its AND predecessors and at least one of its

```
Submethod      decode()
1              Initialise
2              nbMS := 1, ctLeft := 0, ctRight := 0, Count := 0
3              WHILE (count < nbTasks) //until all tasks are assigned
  3.1             Get the next task in the chromosome
  3.2             IF (The side of the task is Left)
    3.2.1            side := 0
  3.3             ELSE IF (The side of the task is Right)
    3.3.1            side := 1
  3.4             ELSE //The side of the task is Either
    3.4.1            Select the side randomly (0 or 1)
  3.5             ENDIF
  3.6             isAssigned := false
  3.7             IF (side = 0) //left side
    3.7.1            es := max(ctLeft, max(ESTOR[task], ESTAND[task]))
    3.7.2            IF ((es + TaskTimes[task]) ≤ CT)
      3.7.2.1           Assign task to the current position
      3.7.2.2           ctLeft := es + TaskTimes[task]
      3.7.2.3           updateEST(task, ctLeft)
      3.7.2.4           isAssigned := true
      3.7.2.5           count++
    3.7.3            ENDIF
  3.8             ELSE IF (side = 1) //right side
    3.8.1            es := max(ctRight, max(ESTOR[task], ESTAND[task]))
    3.8.2            IF ((es + TaskTimes[task]) ≤ CT)
      3.8.2.1           Assign task to the current position
      3.8.2.2           ctRight := es + TaskTimes[task]
      3.8.2.3           updateEST(task, ctRight)
      3.8.2.4           isAssigned := true
      3.8.2.5           count++
    3.8.3            ENDIF
   3.9            ENDIF
   3.10           IF (isAssigned = false) //The task is not assigned
     3.10.1          nbMS++
     3.10.2          Reset ESTOR and ESTAND
     3.10.3          ctLeft := 0, ctRight := 0
   3.11           ENDIF
4              ENDWHILE
```

Fig. 9. The procedure used for decoding chromosomes.

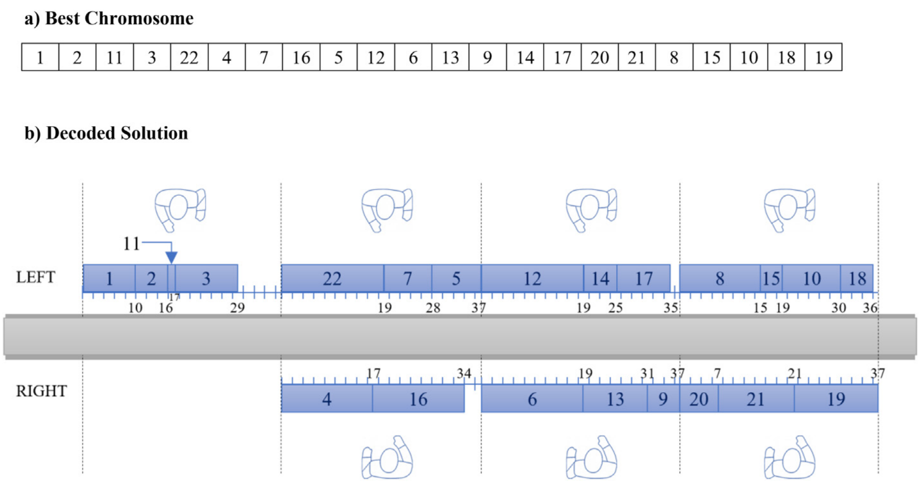


Fig. 10. The best balancing solution obtained using the decoding procedure.

OR successors (if any). As seen from the figure, the infeasible mutant violates the precedence relations in terms of both AND and OR relations. Based on the precedence relations (which was given in Fig. 2), task 10 must be located after at least one of its OR predecessors, i.e. task 6 or task 8. So, it is relocated after the first OR predecessor, task 6, on the chromosome. On the other hand, task 19 must be located after its AND predecessor, namely task 15. Therefore, it is moved to immediately after task 15 on the chromosome. Thus, a feasible chromosome is obtained eventually.

3.5. Forming the new generation

The new generation is formed replacing the worst chromosomes in the population with better individuals (if any) obtained

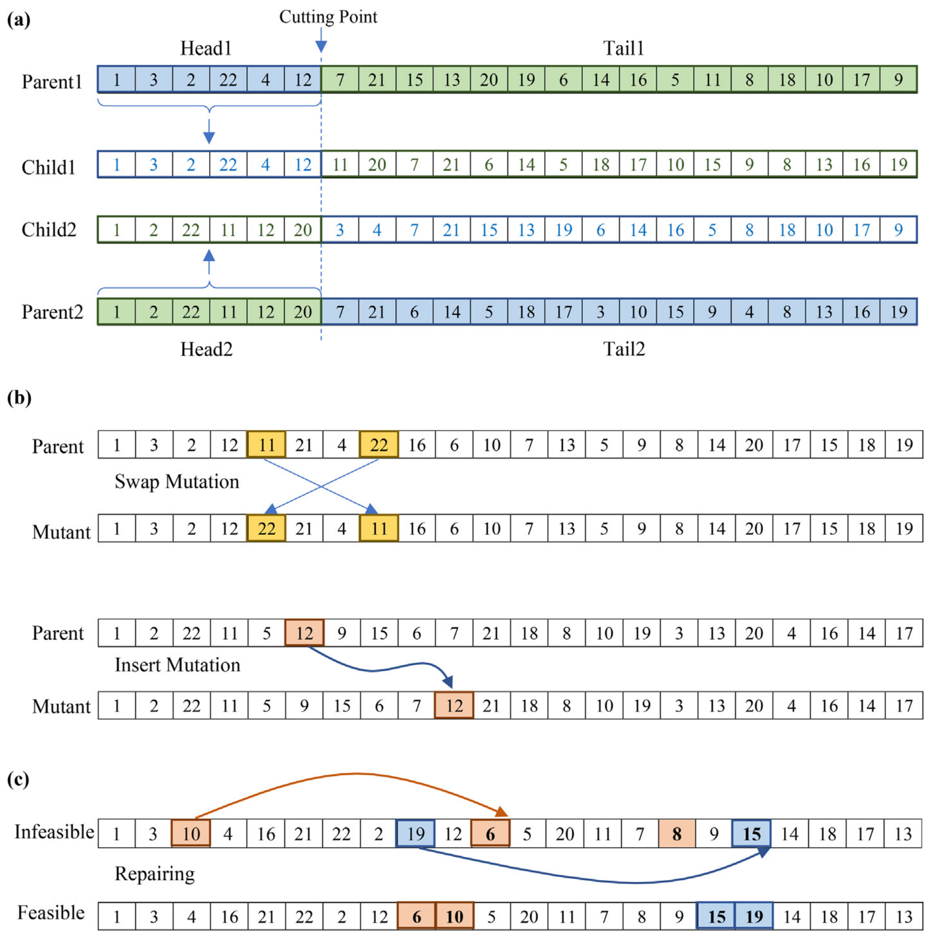


Fig. 11. Genetic operators; (a) Crossover, (b) Mutation and (c) Repairing mechanisms.

during crossover and mutation. Thus, the fitness value of the worst chromosome in the population is compared to that of the best chromosome among the new individuals. If the new individual is better than the worst in the chromosome, the worst is replaced and the second best and second worst individuals are compared. This procedure is repeated until there is no better individual among the children. Note that the chromosome duplications are avoided as duplicated ones are already destroyed after the application of the genetic operators. Replacing the worst policy also helps to preserve the best chromosomes in the population, commonly known as *elitism*.

Fig. 12 presents the convergence of 2-GA for solving the 2P22-OR problem. The best solution (with four mated-stations and seven workstations already presented in Fig. 10) was obtained in the 36th iteration. Therefore, only the first 100 iterations (out of 1000 iterations) are depicted for the purpose of increasing the readability of the graph (note that iteration '0' corresponds to the initial population).

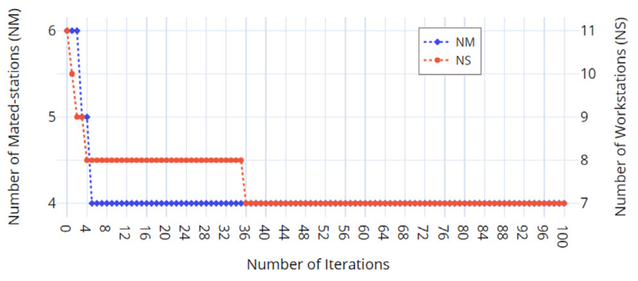


Fig. 12. The convergence of 2-GA through the first 100 iterations.

4. Computational tests

Computational tests have been conducted to observe the performance of the proposed MILP model and the metaheuristic approach, 2-GA. The same problems have also been solved using a tabu search (TS) algorithm for the comparison purpose. The pseudocode of utilised TS algorithm is presented in Appendix-C. The MILP model was solved via CPLEX solver embedded into GAMS. The metaheuristics, 2-GA and TS, were coded in Java using Eclipse IDE v4.7.3a. Experiments have been done on a PC equipped with Intel® Core™ i7-6700HQ CPU @2.60 GHz and 16 GB of RAM.

As the TDLBP has not been studied previously, there was no dataset which can be used directly. Therefore, the test data used for other disassembly line balancing problems in the literature, belonging to various case studies ranging from the disassembly of a ball-point pen to a laptop and a Samsung SCH-3500 cell phone, has been modified and adapted to the specifications of the TDLBP. Specifically, test problems 2P8, 2P10, 2P10-OR, 2P22-OR, 2P25, 2P47 and 2P47-OR have been adapted from P8 (Kalayci and Gupta, 2014), P10 (Kalayci and Gupta, 2013), P10-OR (McGovern and Gupta, 2003), P22-OR (Kalaycılar et al., 2016), P25 (Kalayci and Gupta, 2013), P47 (Kalayci et al., 2015) and P47-OR (Kalaycılar et al., 2016), respectively. Test problems 2P8-OR and 2P25-OR have been modified from 2P8 and 2P25, respectively. Note that 2P47 and 2P47-OR have three sets of operation times, referred to as A, B and C, each of which has been considered as a separate test problem. All data used for the test problems has been provided in Appendix-A for future researches (except for 2P10-OR and 2P22-OR, as these problems have already been presented in Section 2). Test problems have been grouped into three categories (small, medium and large) based on their size.

The solutions obtained by CPLEX, 2-GA and TS under various CT constraints are presented in Tables 1–3 for small, medium and large-size problems, respectively. In Table 1, the column *OPT - NM[NS]* given under CPLEX reports the optimal results within the time reported in the column *CPU*, where *NM* and *NS* denote the number of mated stations and the number of workstations, respectively. The column *Optimality* indicates whether the optimum solution is obtained or not (*Y* and *U* denote *Yes* and *Unknown*, respectively). As the problems in Table 1 are small-sized, the optimal solutions have been obtained by CPLEX for all of them.

The 2-GA and TS were run for five times to solve each test problem and the results are also presented in Table 1. The parameters used for 2-GA and TS are presented in the Appendix-B. Preliminary tests have been conducted to determine the parameter levels based on problem size. The column *BEST-NM[NS]* reports the best solution obtained after five consecutive runs. The average and standard deviation of CPU time consumption (for each run) is also reported in the table in milliseconds – *ms* (see *Avg. CPU* and *StDev CPU*). To measure the general quality of the solutions generated, the column *MAX-NM[NS]* which reports the worst solutions of the five runs, is also included in the table. The columns RPD_{NM} and RPD_{NS} represent the relative percentage deviations of *NM* and *NS* values between the *BEST-NM[NS]* and *MAX-NM[NS]*, calculated as follows.

$$RPD_{NM} = 100(MAX_{NM} - BEST_{NM})/BEST_{NM} \tag{17}$$

$$RPD_{NS} = 100(MAX_{NS} - BEST_{NS})/BEST_{NS} \tag{18}$$

As seen in Table 1 2-GA found optimal solutions for all problems, i.e. #1-#24. As the problems are small scale, the time needed to solve the problems was very small (<10-ms). Therefore, there is no much difference between CPLEX and 2-GA in terms of the CPU time (except some problems for which CPLEX needed 1 s, see for example #20, #22 and #23). Regarding the RPD_{NM} and RPD_{NS} values, it can be stated that the solution building capacity of the 2-GA is quite powerful. With regard to TS, optimum solutions were obtained for all small-size problems except #3, #15 and #23 within<10-ms. For these three problems, TS solutions require one more workstation than that of 2-GA.

For the medium-size problems, the optimal solutions were found by CPLEX only for a small proportion. As the time limit of 1800 s was exceeded in most cases, the optimality of the solutions is not known, except those identified with '*Y*' in its *Optimality* status (see test problems #25-#27, #30-#33, and #48-#49). As seen from Table 2 2-GA was capable of finding the same solutions with CPLEX for 24 out of 28 test problems (including the seven test problems solved optimally). For #26 and #45, the solution found by 2-GA requires one more mated-station than that of CPLEX. For #25, 2-GA found a solution with one more mated-station and workstation in comparison to CPLEX (while only one more workstation for #35). In medium-size problems, the gap between the solutions by 2-GA and TS has increased. TS found the same solutions with CPLEX for 20 out of 28 test problems. 2-GA outperformed TS for problems #26, #34, #36, #37, #39 and #48. One exception is that, TS found a better solution (i.e. 4 (Lambert et al., 2005) than 2-GA (i.e. 5 (Lambert et al., 2005) only for one problem (i.e. #45) for which TS requires one fewer mated station than that of 2-GA. It is clear that 2-GA outperforms TS again for medium-size problems despite slightly higher CPU requirements.

When solving the large-size problems via CPLEX, it was observed that the size of `nodefile` has reached to excessive sizes, almost 4 GB (within 1800 s) in most cases. Therefore, the '`nodefileind`' option was set to '`2`', which enabled CPLEX store information on disk (rather than memory), to overcome the resource limit issues when solving the large-scale problems. Despite these modifications, CPLEX was even unable to provide a feasible solution within the 1800 s time limit for most problems categorised under 2P47-OR. No integer solution was existing when the time limit was exceeded. In that case, only the 2-GA and TS solutions

Table 1
Test problems and their solutions for small-size problems.

#	**Test Problem**	$\sum_{i\in I} t_i$	**CT**	**CPLEX**			**2-GA**						**TS**					
				OPT *NM [NS]*	**CPU (s)**	**Optimality**	**BEST *NM [NS]***	**MAX *NM [NS]***	RPD_{NM}	RPD_{NS}	**Avg. CPU (ms)**	**StDev CPU (ms)**	**BEST *NM [NS]***	**MAX *NM [NS]***	RPD_{NM}	RPD_{NS}	**Avg. CPU (ms)**	**StDev PU (ms)**
1	2P8	149	36	**5[6]**	<1	Y	**5[6]**	5[6]	0	0	<10	<10	**5[6]**	5[6]	0	0	<10	<10
2			37	**5[6]**	<1	Y	**5[6]**	5[6]	0	0	<10	<10	**5[6]**	5[6]	0	0	<10	<10
3			38	**4[4]**	<1	Y	**4[4]**	4[5]	0	0.25	<10	<10	4[5]	4[5]	0	0	<10	<10
4			39	**4[4]**	<1	Y	**4[4]**	4[4]	0	0.25	<10	<10	**4[4]**	4[4]	0	0	<10	<10
5			40	**4[4]**	<1	Y	**4[4]**	4[4]	0	0	<10	<10	**4[4]**	4[4]	0	0	<10	<10
6			41	**4[4]**	<1	Y	**4[4]**	4[5]	0	0.25	<10	<10	**4[4]**	4[4]	0	0	<10	<10
7	2P10	169	36	**4[5]**	<1	Y	**4[5]**	4[5]	0	0	<10	<10	**4[5]**	4[6]	0	0.2	<10	<10
8			39	**4[5]**	<1	Y	**4[5]**	4[5]	0	0	<10	<10	**4[5]**	4[6]	0	0.2	<10	<10
9			42	**3[5]**	<1	Y	**3[5]**	3[5]	0	0	<10	<10	**3[5]**	4[5]	0.33	0	<10	<10
10			44	**3[5]**	<1	Y	**3[5]**	3[5]	0	0	<10	<10	**3[5]**	4[5]	0.33	0	<10	<10
11			46	**3[4]**	<1	Y	**3[4]**	4[5]	0.33	0.25	<10	<10	**3[4]**	4[5]	0.33	0.25	<10	<10
12			48	**2[4]**	<1	Y	**2[4]**	2[4]	0	0	<10	<10	**2[4]**	3[5]	0.5	0.25	<10	<10
13	2P8-OR	149	36	**5[5]**	<1	Y	**5[5]**	5[5]	0	0	<10	<10	**5[5]**	5[6]	0	0.2	<10	<10
14			37	**4[5]**	<1	Y	**4[5]**	4[5]	0	0	<10	<10	**4[5]**	5[6]	0.25	0.2	<10	<10
15			38	**3[4]**	<1	Y	**3[4]**	3[4]	0	0	<10	<10	3[5]	4[5]	0.33	0	<10	<10
16			39	**3[4]**	<1	Y	**3[4]**	3[5]	0	0.25	<10	<10	**3[4]**	4[4]	0.33	0	<10	<10
17			40	**3[4]**	<1	Y	**3[4]**	3[4]	0	0	<10	<10	**3[4]**	4[4]	0.33	0	<10	<10
18			41	**3[4]**	<1	Y	**3[4]**	3[4]	0	0	<10	<10	**3[4]**	4[4]	0.33	0	<10	<10
19	2P10-OR	173	38	**4[5]**	<1	Y	**4[5]**	4[5]	0	0	<10	<10	**4[5]**	4[6]	0	0.2	<10	<10
20			40	**4[5]**	1	Y	**4[5]**	4[5]	0	0	<10	<10	**4[5]**	4[6]	0	0.2	<10	<10
21			43	**3[5]**	<1	Y	**3[5]**	3[5]	0	0	<10	<10	**3[5]**	4[6]	0.33	0.2	<10	<10
22			45	**3[5]**	1	Y	**3[5]**	3[5]	0	0	<10	<10	**3[5]**	4[5]	0.33	0	<10	<10
23			46	**3[4]**	1	Y	**3[4]**	3[5]	0	0.25	<10	<10	3[5]	3[5]	0	0	<10	<10
24			48	**2[4]**	<1	Y	**2[4]**	2[4]	0	0	<10	<10	**2[4]**	3[5]	0.5	0.25	<10	<10

*Best in bold.

Table 2
Test problems and their solutions for medium-size problems.

#	Test Problem	$\sum_{i \in I} t_i$	CT	CPLEX			2-GA						TS					
				NM [NS]	CPU (s)	Optimality	BEST *NM [NS]*	MAX *NM [NS]*	RPD_{NM}	RPD_{NS}	Avg. CPU (ms)	StDev CPU (ms)	BEST *NM [NS]*	MAX *NM [NS]*	RPD_{NM}	RPD_{NS}	Avg. CPU (ms)	StDev CPU (ms)
25	2P25	155	18	**5[9]**	110	Y	6[10]	6[10]	0	0	67	27	5[10]	7[11]	0.4	0.1	46	13
26			20	**4[8]**	139	Y	5[8]	5[9]	0	0.125	68	27	5[9]	5[10]	0	0.111	52	15
27			22	**4[8]**	162	Y	**4[8]**	4[8]	0	0	71	34	**4[8]**	5[9]	0.25	0.125	49	15
28			24	**4[7]**	1800	U	**4[7]**	4[8]	0	0.142	95	44	**4[7]**	5[9]	0.25	0.285	63	25
29			26	**4[7]**	1800	U	**4[7]**	4[7]	0	0	60	33	**4[7]**	4[8]	0	0.142	45	16
30			28	**3[6]**	32	Y	**3[6]**	3[6]	0	0	80	40	**3[6]**	4[7]	0.333	0.166	47	16
31			30	**3[6]**	977	Y	**3[6]**	3[6]	0	0	64	28	**3[6]**	4[7]	0.333	0.166	85	24
32			32	**3[5]**	64	Y	**3[5]**	3[6]	0	0.2	86	51	**3[5]**	3[6]	0	0.2	49	23
33			34	**3[5]**	334	Y	**3[5]**	3[5]	0	0	70	35	**3[5]**	3[6]	0	0.2	45	17
34	2P22-OR	245	26	**6[10]**	1800	U	**6[10]**	6[11]	0	0.1	66	35	6[11]	8[13]	0.333	0.181	34	16
35			28	**6[9]**	1800	U	6[10]	6[10]	0	0	99	25	6[10]	7[12]	0.166	0.2	31	21
36			30	**5[9]**	1800	U	**5[9]**	6[9]	0.2	0	64	39	6[10]	7[11]	0.166	0.1	37	12
37			32	**5[8]**	1800	U	**5[8]**	5[8]	0	0	98	61	5[9]	6[10]	0.2	0.111	35	12
38			34	**5[8]**	1800	U	**5[8]**	5[9]	0	0.125	54	36	**5[8]**	6[9]	0.2	0.125	31	13
39			36	**4[7]**	1800	U	**4[7]**	5[7]	0.25	0	99	31	5[7]	5[10]	0	0.428	25	9
40			38	**4[7]**	1800	U	**4[7]**	4[7]	0	0	92	37	**4[7]**	5[8]	0.25	0.142	33	14
41			40	**4[7]**	1800	U	**4[7]**	4[7]	0	0	96	40	**4[7]**	5[9]	0.25	0.285	39	16
42			42	**4[7]**	1800	U	**4[7]**	4[7]	0	0	87	37	**4[7]**	4[8]	0	0.142	77	20
43			44	**4[6]**	1800	U	**4[6]**	4[6]	0	0	70	36	**4[6]**	4[8]	0	0.333	29	12
44	2P25-OR	155	19	**5[9]**	1800	U	**5[9]**	5[10]	0	0.111	88	49	**5[9]**	7[11]	0.4	0.222	53	19
45			21	**4[8]**	1800	U	5[8]	5[8]	0	0	71	37	**4[8]**	6[10]	0.5	0.25	51	24
46			23	**4[8]**	1800	U	**4[8]**	5[8]	0.25	0	78	50	**4[8]**	5[9]	0.25	0.125	48	17
47			25	**4[7]**	1800	U	**4[7]**	4[7]	0	0	79	25	**4[7]**	5[8]	0.25	0.142	44	13
48			27	**3[6]**	62	Y	**3[6]**	4[6]	0.333	0	84	50	4[6]	4[7]	0	0.166	46	15
49			29	**3[6]**	35	Y	**3[6]**	3[6]	0	0	73	33	**3[6]**	4[7]	0.333	0.166	38	13
50			31	**3[6]**	1800	U	**3[6]**	3[6]	0	0	65	34	**3[6]**	3[6]	0	0	42	12
51			33	**3[5]**	1800	U	**3[5]**	3[5]	0	0	75	35	**3[5]**	3[6]	0	0.2	37	12
52			35	**3[5]**	1800	U	**3[5]**	3[5]	0	0	67	32	**3[5]**	3[6]	0	0.2	41	11

*Best in bold.

have been provided for some instances of test problems 2P47-OR-A, 2P47-OR-B and 2P47-OR-C.

From #53 to #70, CPLEX solutions were presented for 18 test problems, among which 2-GA was able to provide six better solutions than CPLEX (see test problems #53, #62, #63, #65, #69 and #70) and 12 better solutions than TS (i.e. #53-#55, #58-#60, #62-#63 and #65-#68). The results obtained by 2-GA were the same with CPLEX for the 12 problems, and with TS for the six test problems. From #71 to #88 (where the OR relations were considered), 2-GA has clearly outperformed the CPLEX in all instances. CPLEX was able to come up with a feasible solution for test problems #72, #76 and #86. However, these results were also surpassed by those obtained by 2-GA. The relative percentage deviations, presented in columns RPD_{NM} and RPD_{NS}, also indicate that the algorithm is quite steady and effective. With regard to OR relational problems (#71-#88) TS performed worse than 2-GA for 11 problems, i.e. #71, #75-#78, #81-#85 and #87. With the cost of moderately higher CPU times, 2-GA also outperforms TS for large-size problems with lower RPD_{NM} and RPD_{Ns} values. However, the CPU times required by 2-GA are still dramatically lower than that required by CPLEX. While CPLEX fell short of finding even a feasible solution for most of the test problems of large-size within the time limit, 2-GA was very fast and efficient in finding sufficient quality results within a very short period of time.

5. Conclusions and future research

Almost three decades have passed since the DLBP was first brought to the attention of the academia. However, no research was conducted on two-sided disassembly lines since then. This research introduced the TDLBP and defined it providing many challenging issues. The introduced TDLBP was also modelled mathematically via a novel MILP model, which can be used for solving small and some of the medium-size TDLBPs optimally. A powerful metaheuristic, called 2-GA, was developed considering the complex precedence relationship nature of the TDLBPs. The solution building and reproduction mechanisms of the proposed 2-GA have been described in detail and demonstrated via numerical examples.

As there is no research published concerning this new problem type, a set of 88 test instances has been newly generated modifying existing data for other problem types. Computational tests have been conducted using the newly generated data to test the performance of 2-GA. The test problems have been solved using both proposed MILP model (via CPLEX), 2-GA and TS and the results were compared. The comparison made it clear that 2-GA has a superior and steady performance in terms of both the quality of the solutions and the computational time required. The optimal solutions of the problems have been obtained for all small-size problems. For the majority of the medium-size problems, specifically 24 out of 28 test problems, 2-GA obtained the same results with CPLEX. As expected, 2-GA outperformed the CPLEX when solving the large-size problems, where the search space grows dramatically with the increasing number of tasks. Furthermore, 2-GA also performed better than TS for some of the medium-size problems and majority of the large-size problems.

In practice, the proposed 2-GA can be effectively used when the problem size exceeds the capability of the proposed MILP model. As it generates robust and high-quality solutions in a very short

Table 3
Test problems and their solutions for large-size problems.

#	Test Problem	$\sum_{i\in I} t_i$	CT	CPLEX			2-GA						TS					
				NM [NS]	CPU (s)	Optimality	BEST *NM [NS]*	MAX *NM [NS]*	RPD_{NM}	RPD_{NS}	Avg. CPU (ms)	StDev CPU (ms)	BEST *NM [NS]*	MAX *NM [NS]*	RPD_{NM}	RPD_{NS}	Avg. CPU (ms)	StDev CPU (ms)
53	2P47-A	712	98	5[9]	1800	U	**5[8]**	5[9]	0	0.125	634	84	5[9]	6[11]	0.2	0.222	421	45
54			101	**4[8]**	1800	U	**4[8]**	5[8]	0.25	0	636	57	5[8]	6[11]	0.2	0.375	485	43
55			104	**4[8]**	1800	U	**4[8]**	4[8]	0	0	666	69	5[9]	6[10]	0.2	0.111	549	51
56			107	**4[8]**	1800	U	**4[8]**	4[8]	0	0	815	181	**4[8]**	6[10]	0.5	0.25	631	134
57			110	**4[8]**	1800	U	**4[8]**	4[8]	0	0	722	122	**4[8]**	6[9]	0.5	0.125	765	149
58			113	**4[7]**	1800	U	**4[7]**	4[7]	0	0	980	98	4[8]	5[10]	0.25	0.25	648	93
59	2P47-B	856	104	**5[9]**	1800	U	**5[9]**	6[10]	0.20	0.111	829	97	6[10]	7[12]	0.166	0.2	822	117
60			108	**5[9]**	1800	U	**5[9]**	5[9]	0	0	728	100	5[10]	7[12]	0.4	0.2	748	99
61			112	**5[9]**	1800	U	**5[9]**	5[9]	0	0	946	154	**5[9]**	6[11]	0.2	0.222	620	82
62			116	5[8]	1800	U	**4[8]**	5[10]	0.25	0.25	1163	172	5[8]	6[11]	0.2	0.375	834	119
63			120	5[8]	1800	U	**4[8]**	5[8]	0.25	0	856	223	5[9]	6[11]	0.2	0.222	959	166
64			124	**4[8]**	1800	U	**4[8]**	5[8]	0.25	0	823	64	**4[8]**	6[10]	0.5	0.25	649	173
65	2P47-C	1045	110	6[12]	1800	U	**6[11]**	7[11]	0.166	0	929	147	7[12]	8[14]	0.142	0.166	558	95
66			115	**6[10]**	1800	U	**6[10]**	7[11]	0.166	0.1	728	152	6[11]	8[13]	0.333	0.181	645	138
67			120	**6[10]**	1800	U	**6[10]**	6[10]	0	0	87	179	6[11]	7[14]	0.166	0.272	814	118
68			125	**5[9]**	1800	U	**5[9]**	6[10]	0.20	0.111	880	217	6[10]	7[12]	0.166	0.2	628	107
69			130	6[10]	1800	U	**5[9]**	5[9]	0	0	879	278	**5[9]**	7[12]	0.4	0.333	726	117
70			135	6[10]	1800	U	**5[9]**	5[9]	0	0	917	187	**5[9]**	6[12]	0.2	0.333	846	127
71	2P47-OR-A	712	99	-	1800	U	**4[8]**	5[9]	0.25	0.125	1212	444	5[9]	6[10]	0.2	0.111	689	143
72			102	5[9]	1800	U	**4[8]**	5[9]	0.25	0.125	1732	336	**4[8]**	6[10]	0.5	0.25	954	243
73			105	-	1800	U	**4[8]**	4[8]	0	0	1589	316	**4[8]**	5[9]	0.25	0.125	1086	523
74			108	-	1800	U	**4[8]**	4[8]	0	0	1433	483	**4[8]**	5[9]	0.25	0.125	798	412
75			111	-	1800	U	**4[7]**	4[8]	0	0.142	1833	441	4[8]	5[9]	0.25	0.125	1204	325
76			114	5[9]	1800	U	**4[7]**	4[8]	0	0.142	1617	383	4[8]	5[9]	0.25	0.125	727	262
77	2P47-OR-B	856	105	-	1800	U	**5[9]**	5[10]	0	0.111	1571	321	5[10]	6[11]	0.2	0.1	864	167
78			109	-	1800	U	**5[9]**	5[9]	0	0	2305	344	5[10]	6[11]	0.2	0.1	1054	221
79			113	-	1800	U	**5[9]**	5[9]	0	0	1597	486	**5[9]**	6[11]	0.2	0.222	955	147
80			117	-	1800	U	**5[9]**	5[9]	0	0	1981	627	**5[9]**	6[10]	0.2	0.111	889	150
81			121	-	1800	U	**4[8]**	5[9]	0.25	0.125	1865	578	5[9]	6[10]	0.2	0.111	984	214
82			125	-	1800	U	**4[8]**	4[8]	0	0	1827	354	5[9]	5[10]	0	0.111	805	399
83	2P47-OR-C	1045	112	-	1800	U	**6[10]**	6[11]	0	0.1	1942	265	6[11]	7[13]	0.166	0.181	997	129
84			120	-	1800	U	**5[10]**	6[10]	0.20	0	2027	438	6[10]	7[12]	0.166	0.2	1173	208
85			128	-	1800	U	**5[9]**	5[10]	0	0.111	1464	356	5[10]	6[11]	0.2	0.1	1261	218
86			136	5[10]	1800	U	**5[9]**	5[9]	0	0	1575	181	**5[9]**	6[11]	0.2	0.222	761	130
87			144	-	1800	U	**4[8]**	5[9]	0.25	0.125	1682	396	5[9]	5[10]	0	0.111	1037	344
88			152	-	1800	U	**4[8]**	4[8]	0	0	2068	438	**4[8]**	5[10]	0.25	0.25	847	387

*Best in bold.

period of time, the decision makers (i.e. line managers in real world) can effectively make use of it to minimise the length of the line and increase their line efficiency via minimising the number of workstations/operators.

The MILP model and 2-GA approach proposed in this study can easily be extended to adapt other line configurations, such as U-shaped disassembly lines and parallel two-sided disassembly lines (building work over Hezer and Kara (2015)). The disassembly of more than one product model on the same two-sided disassembly line can be of a very interesting research area for future studies. Moreover, new metaheuristics can be developed for solving the introduced TDLBP and their performances can be compared to 2-GA presented in that work. Finally, the short-term storage of removed parts in the workstations can be considered and a time and space DLBP can be introduced in future works.

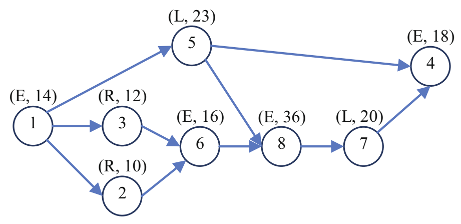
CRediT authorship contribution statement

Ibrahim Kucukkoc: Conceptualization, Methodology, Software, Data curation, Writing - original draft, Visualization, Investigation, Validation, Writing - review & editing.

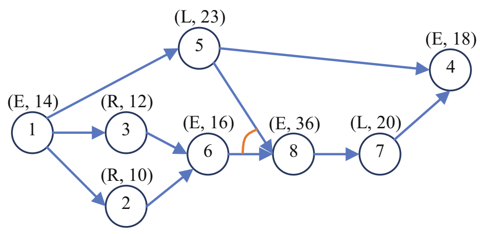
Appendix. -A

Data used for computational tests

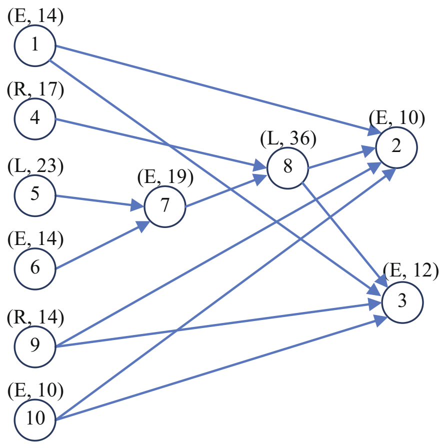
2P8



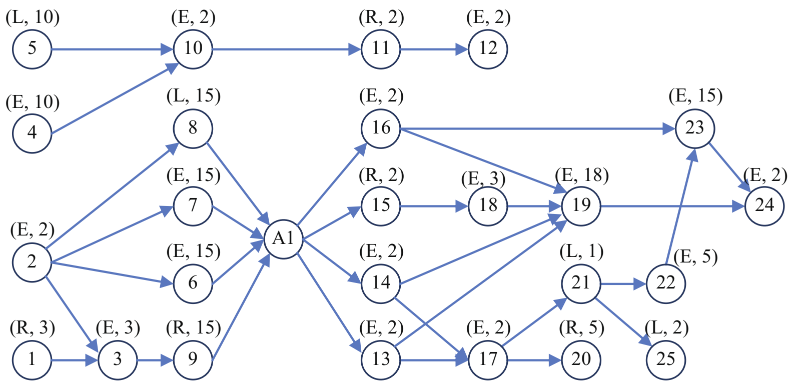
2P8-OR



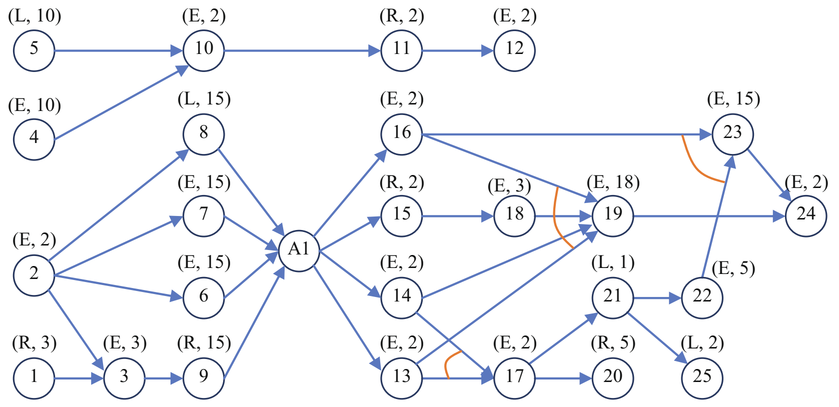
2P10



2P25



2P25-OR



2P47

Part	Side	Time A	Time B	Time C	AND Predecessor(s)
1	E	14	16	18	–
2	L	28	32	36	1
3	E	3	4	9	2
4	R	2	4	6	2
5	L	3	4	9	–
6	R	4	6	8	5
7	E	8	10	12	–
8	E	12	16	20	–
9	E	4	5	6	8
10	E	28	32	36	8
11	L	3	4	9	9, 10
12	E	4	6	8	10
13	L	6	8	10	–
14	E	1	2	3	13
15	E	20	24	28	–
16	R	5	7	9	15
17	E	28	32	36	16
18	L	4	6	8	17
19	E	3	4	9	14
20	R	12	16	20	18
21	E	3	4	9	20
22	E	3	4	9	18
23	R	28	32	36	22
24	E	28	32	36	22
25	L	12	16	20	23,24
26	E	76	88	100	19
27	R	6	8	10	26
28	E	28	32	36	27
29	E	3	6	11	28
30	R	6	8	10	29
31	L	3	4	9	30
32	E	98	104	110	1,5,9,18
33	R	14	18	22	32
34	E	2	4	6	33
35	E	6	8	10	33
36	E	7	8	12	34,35
37	L	60	72	84	36
38	E	6	8	10	37
39	E	60	72	84	38
40	E	8	10	12	39
41	R	12	16	20	40
42	E	3	4	9	41
43	R	12	16	20	36
44	E	3	4	9	43
45	E	28	32	36	44
46	E	2	4	6	45
47	E	3	4	9	18

2P47-OR

Part	Side	Time A	Time B	Time C	AND Predecessor(s)	OR Predecessor(s)
1	E	14	16	18	–	–
2	L	28	32	36	1	–
3	E	3	4	9	1	–
4	R	2	4	6	2	–
5	L	3	4	9	–	4, 11
6	R	4	6	8	32	–
7	E	8	10	12	6, 42	–
8	E	12	16	20	34	–

Appendix (*continued*)

Part	Side	Time A	Time B	Time C	AND Predecessor(s)	OR Predecessor(s)
9	E	4	5	6	8	–
10	E	28	32	36	45	9, 30
11	L	3	4	9	3	–
12	E	4	6	8	3	–
13	L	6	8	10	12	–
14	E	1	2	3	12	–
15	E	20	24	28	12	–
16	R	5	7	9	32	–
17	E	28	32	36	6	–
18	L	4	6	8	34	–
19	E	3	4	9	14	–
20	R	12	16	20	15	–
21	E	3	4	9	15	–
22	E	3	4	9	14	–
23	R	28	32	36	33	–
24	E	28	32	36	33	–
25	L	12	16	20	–	19, 21
26	E	76	88	100	20	–
27	R	6	8	10	20	–
28	E	28	32	36	–	24, 25, 26
29	E	3	6	11	17, 27	–
30	R	6	8	10	–	18, 28, 29
31	L	3	4	9	–	3, 4
32	E	98	104	110	–	5, 13
33	R	14	18	22	–	16, 22
34	E	2	4	6	–	7, 23
35	E	6	8	10	5	–
36	E	7	8	12	35	–
37	L	60	72	84	35	–
38	E	6	8	10	35	–
39	E	60	72	84	36	–
40	E	8	10	12	36	–
41	R	12	16	20	–	37, 39
42	E	3	4	9	38	–
43	R	12	16	20	38	-
44	E	3	4	9	–	38, 40, 41
45	E	28	32	36	–	37, 42
46	E	2	4	6	–	40, 43
47	E	3	4	9	44	–

Appendix. -B

The parameters used by 2-GA and TS

	2-GA				TS	
Problem Set	popSize	CR	MR	maxIter	tabuSize	maxIter
2P8	8	0.4	0.1	200	16	400
2P10	10	0.4	0.2	200	20	400
2P8-OR	20	0.4	0.2	400	16	800
2P10-OR	20	0.4	0.2	400	20	800
2P-25	20	0.6	0.2	800	50	1600
2P22-OR	20	0.6	0.2	2000	44	4000
2P25-OR	20	0.6	0.2	2000	50	4000
2P47	50	0.6	0.2	5000	94	10000
2P47-OR	50	0.6	0.2	5000	94	10000

Appendix. -C

The pseudocode of TS algorithm

```
Algorithm     TS()
1             chrom := buildChrom()
2             chromFitness := decode(chrom)
3             iter := 1
4             WHILE (iter ≤ maxIter) //until a termination criterion is satisfied
  4.1             r1 := random(1,nbTasks), r2 := random(1,nbTasks)
  4.2             tabu := checkTabu(r1,r2)
  4.3             WHILE (tabu=true or r1=r2) //until the move is not tabu
    4.3.1             r1 := random(1,nbTasks), r2 := random(1,nbTasks)
    4.3.2             tabu := checkTabu(r1,r2)
  4.4             ENDWHILE
  4.5             neigh := chrom
  4.6             movedTask := r1th task on neigh
  4.7             Remove movedTask from neigh
  4.8             Add movedTask to r2th position on neigh
  4.9             Repair neigh
  4.10            tabu1[movedTask][r2] := iter + tabuSize; //update TabuTable1
  4.11            tabu2[r2][r1] := iter + 2;               //update TabuTable2
  4.12            neighFitness := decode(neigh)
  4.13            IF (neighFitness < chromFitness) //if a new solution is better
    4.13.1            chrom := neigh
    4.13.2            chromFitness := neighFitness
  4.14            ENDIF
  4.15            iter := iter+1
5             ENDWHILE
```

```
Submethod     checkTabu(r1,r2)
1             IF (tabu1[movedTask][r2] ≤ iter  and  tabu2[r2][r1] ≤ iter)
 1.1              tabu := false
2             ELSE
 2.1              tabu := true
3             END
4             return tabu
```

References

Agrawal, S., Tiwari, M.K., 2008. A collaborative ant colony algorithm to stochastic mixed-model U-shaped disassembly line balancing and sequencing problem. Int. J. Prod. Res. 46 (6), 1405–1429.
Altekin, F.T., Kandiller, L., Ozdemirel, N.E., 2008. Profit-oriented disassembly-line balancing. Int. J. Prod. Res. 46 (10), 2675–2693.
Altekin, F.T., Akkan, C., 2012. Task-failure-driven rebalancing of disassembly lines. Int. J. Prod. Res. 50 (18), 4955–4976.
Bartholdi, J.J., 1993. Balancing 2-sided assembly lines - a case-study. Int. J. Prod. Res. 31 (10), 2447–2461.
Bentaha, M.L., Battaïa, O., Dolgui, A., 2015. An exact solution approach for disassembly line balancing problem under uncertainty of the task processing times. Int. J. Prod. Res. 53 (6), 1807–1818.
Ding, L.-P. et al., 2010. A new multi-objective ant colony algorithm for solving the disassembly line balancing problem. Int. J. Adv. Manuf. Technol. 48 (5), 761–771.
Gen, M., Altiparmak, F., Lin, L., 2006. A genetic algorithm for two-stage transportation problem using priority-based encoding. OR Spectrum 28 (3), 337–354.
Güngör, A., Gupta, S.M., 1999. Disassembly line balancing. In: Proceedings of the Annual Meeting of the Northeast Decision Sciences Institute, Newport, RI..
Gungor, A., Gupta, S.M., 2001. A solution approach to the disassembly line balancing problem in the presence of task failures. Int. J. Prod. Res. 39 (7), 1427–1467.
Güngör, A., Gupta, S.M., 2002. Disassembly line in product recovery. Int. J. Prod. Res. 40 (11), 2569–2589.
Hezer, S., Kara, Y., 2015. A network-based shortest route model for parallel disassembly line balancing problem. Int. J. Prod. Res. 53 (6), 1849–1865.
Kalayci, C.B., Gupta, S.M., 2013. A particle swarm optimization algorithm with neighborhood-based mutation for sequence-dependent disassembly line balancing problem. Int. J. Adv. Manuf. Technol. 69 (1), 197–209.
Kalayci, C.B., Gupta, S.M., 2013. Artificial bee colony algorithm for solving sequence-dependent disassembly line balancing problem. Expert Syst. Appl. 40 (18), 7231–7241.
Kalayci, C.B., Gupta, S.M., 2013. Ant colony optimization for sequence-dependent disassembly line balancing problem. J. Manuf. Technol. Manag. 24 (3), 413–427.
Kalayci, C.B., Polat, O., Gupta, S.M., 2016. A hybrid genetic algorithm for sequence-dependent disassembly line balancing problem. Ann. Oper. Res. 242 (2), 321–354.
Kalayci, C.B., Gupta, S.M., 2014. A tabu search algorithm for balancing a sequence-dependent disassembly line. Prod. Plann. Control 25 (2), 149–160.
Kalayci, C.B. et al., 2015. Multi-objective fuzzy disassembly line balancing using a hybrid discrete artificial bee colony algorithm. J. Manuf. Syst. 37, 672–682.
Kalaycılar, E.G., Azizoğlu, M., Yeralan, S., 2016. A disassembly line balancing problem with fixed number of workstations. Eur. J. Oper. Res. 249 (2), 592–604.
Koc, A., Sabuncuoglu, I., Erel, E., 2009. Two exact formulations for disassembly line balancing problems with task precedence diagram construction using an AND/OR graph. IIE Trans. 41 (10), 866–881.
Kucukkoc, I., Zhang, D.Z., 2015a. Type-E parallel two-sided assembly line balancing problem: mathematical model and ant colony optimisation based approach with optimised parameters. Comput. Ind. Eng. 84, 56–69. https://doi.org/10.1016/j.cie.2014.12.037.

Kucukkoc, I., Zhang, D.Z., 2015b. A mathematical model and genetic algorithm-based approach for parallel two-sided assembly line balancing problem. Prod. Plann. Control 26 (11), 874–894. https://doi.org/10.1080/09537287.2014.994685.

Kucukkoc, I., Zhang, D.Z., 2015c. Integrating ant colony and genetic algorithms in the balancing and scheduling of complex assembly lines. Int. J. Adv. Manuf. Technol. 82 (1), 265–285. https://doi.org/10.1007/s00170-015-7320-y.

Kucukkoc, I., Li, Z., Li, Y., 2020. Type-E disassembly line balancing problem with multi-manned workstations. Optimization Eng. 21 (2), 611–630.

Lambert, A.J.D., Gupta, S.M., 2005. Disassembly Modeling for Assembly, Maintenance, Reuse, and Recycling. CRC Press, Boca Raton, FL.

Li, Z., Tang, Q., Zhang, L., 2017. Two-sided assembly line balancing problem of type I: Improvements, a simple algorithm and a comprehensive study. Comput. Oper. Res. 79, 78–93.

Li, Z., Kucukkoc, I., Zhang, Z., 2020. Branch, bound and remember algorithm for two-sided assembly line balancing problem. Eur. J. Oper. Res. 284 (3), 896–905. https://doi.org/10.1016/j.ejor.2020.01.032.

Li, Z. et al., 2019. A fast branch, bound and remember algorithm for disassembly line balancing problem. Int. J. Prod. Res., 1–15.

Liu, J. et al., 2018. An improved multi-objective discrete bees algorithm for robotic disassembly line balancing problem in remanufacturing. Int. J. Adv. Manuf. Technol. 97 (9–12), 3937–3962.

Liu, J., Wang, S., 2017. Balancing disassembly line in product recovery to promote the coordinated development of economy and environment. Sustainability 9 (2), 309.

Make, A.M.R., Rashid, M.F.F. Ab., Razali, M.M., 2016. A review of two-sided assembly line balancing problem. Int. J. Adv. Manuf. Technol. 89, 1743–1763.

McGovern, S., Gupta, S., 2003. 2-opt heuristic for the disassembly line balancing problem. In: Proceedings of the SPIE International Conference on Environmentally Conscious Manufacturing III, Providence, RI2003, pp. 71–84..

McGovern, S.M., Gupta, S.M., 2006. Ant colony optimization for disassembly sequencing with multiple objectives. Int. J. Adv. Manuf. Technol. 30 (5), 481–496.

McGovern, S.M., Gupta, S.M., 2007. A balancing method and genetic algorithm for disassembly line balancing. Eur. J. Oper. Res. 179 (3), 692–708.

McGovern, S.M., Gupta, S.M., 2007. Combinatorial optimization analysis of the unary NP-complete disassembly line balancing problem. Int. J. Prod. Res. 45 (18–19), 4485–4511.

Mete, S. et al., 2016. Resource constrained disassembly line balancing problem. IFAC-PapersOnLine 49 (12), 921–925.

Mete, S. et al., 2018. An optimisation support for the design of hybrid production lines including assembly and disassembly tasks. Int. J. Prod. Res., 1–15.

Mete, S. et al., 2016. A solution approach based on beam search algorithm for disassembly line balancing problem. J. Manuf. Syst. 41, 188–200.

Özceylan, E. et al., 2018. Disassembly line balancing problem: a review of the state of the art and future directions. Int. J. Prod. Res., 1–23.

Paksoy, T. et al., 2013. Mixed model disassembly line balancing problem with fuzzy goals. Int. J. Prod. Res. 51 (20), 6082–6096.

Ren, Y. et al., 2017. An improved gravitational search algorithm for profit-oriented partial disassembly line balancing problem. Int. J. Prod. Res. 55 (24), 7302–7316.

Ren, Y. et al., 2018. An asynchronous parallel disassembly planning based on genetic algorithm. Eur. J. Oper. Res. 269 (2), 647–660.

Ren, Y. et al., 2018. Disassembly line balancing problem using interdependent weights-based multi-criteria decision making and 2-Optimal algorithm. J. Cleaner Prod. 174, 1475–1486.

Ren, Y. et al., 2018. An MCDM-based multiobjective general variable neighborhood search approach for disassembly line balancing problem. IEEE Trans. Syst. Man Cybernet. Syst., 1–14.

Sadegheih, A., 2006. Scheduling problem using genetic algorithm, simulated annealing and the effects of parameter values on GA performance. Appl. Math. Model. 30 (2), 147–154.

Vignaux, G.A., Michalewicz, Z., 1991. A genetic algorithm for the linear transportation problem. IEEE Trans. Syst. Man Cybernet. 21 (2), 445–452.

Xiao, S. et al., 2017. An entropy-based adaptive hybrid particle swarm optimization for disassembly line balancing problems. Entropy 19 (11), 596.

Zhang, Z. et al., 2017. A Pareto improved artificial fish swarm algorithm for solving a multi-objective fuzzy disassembly line balancing problem. Expert Syst. Appl. 86, 165–176.

Zhu, L., Zhang, Z., Wang, Y., 2018. A Pareto firefly algorithm for multi-objective disassembly line balancing problems with hazard evaluation. Int. J. Prod. Res. 56 (24), 7354–7374.