Robotic disassembly re-planning using a two-pointer detection strategy and a super-fast bees algorithm

Laili, Yuanjun; Tao, Fei; Pham, Duc; Wang, Yongjing; Zhang, Lin

DOI:
10.1016/j.rcim.2019.04.003

Document Version
Peer reviewed version

Citation for published version (Harvard):
Laili, Y, Tao, F, Pham, D, Wang, Y & Zhang, L 2019, 'Robotic disassembly re-planning using a two-pointer detection strategy and a super-fast bees algorithm', *Robotics and Computer-Integrated Manufacturing*, vol. 59, pp. 130-142. https://doi.org/10.1016/j.rcim.2019.04.003

Robotic disassembly re-planning using a two-pointer detection strategy and a super-fast bees algorithm

Yuanjun Laili[1], Fei Tao[1*], Duc Truong Pham[2], Yongjing Wang[2], Lin Zhang[1]

[1]*School of Automation Science and Electrical Engineering, Beihang University, Beijing, 100191, China*

[2]*School of Engineering, University of Birmingham, Birmingham, B15 2TT, UK*

[*]*(Corresponding author: Fei Tao, Email: ftao@buaa.edu.cn)*

Abstract: Automated disassembly of End-of-Life (EoL) products can be difficult to implement due to uncertainties in their conditions. An automatic re-planning function is required to enable flexible adjustments of disassembly plans and thus increase disassembly efficiency. The re-planning function is able to detect subassemblies and separable components, and adjust disassembly sequences and directions when components interlock and are irremovable. This paper presents a two-pointer detection strategy to find detachable subassemblies very quickly. A summation operator and a list with two pointers are used to check the interferences between components in a minimum number of steps. Then, a ternary bees algorithm is proposed to identify new disassembly sequences and directions. The algorithm combines the merits of a greedy search and meta-heuristic techniques by using only three collaborative potential solutions and three concurrent operations. Experimental results show that the proposed approach is able to perform a rapid subassembly detection and sequence optimisation for a robotic disassembly task, thus allowing real-time re-planning.

Key words: Robotic disassembly; Component and subassembly detection; Sequence planning; Bees algorithm

1 Introduction

Disassembly of End-of-Life (EoL) products to retrieve reusable parts and materials enables industry to make more efficient use of limited natural resources and reduce environmental pollution [1][2]. Unlike new product assembly, disassembly of EoL products is affected by significant uncertainties mainly caused by structure degrading [3]. Components can be worn-out, corroded, or even missing, and this

creates many unforeseen EoL conditions. These uncertainties can cause frequent changes of disassembly plans, and therefore, reduce efficiency.

The focus of previous research has primarily been on three key areas, namely, disassembly scheduling, disassembly sequencing and disassembly line balancing, each aimed at improving the speed and efficiency of disassembly. Disassembly scheduling is "the problem of determining the quantity and timing of the end-of-use/life products while satisfying the demand of their parts over a planning horizon" [4]. Disassembly sequencing involves searching for an optimum order to dismantle a product partially or completely [5]. Disassembly line balancing is to assign multiple disassembly tasks to a group of workstations on a disassembly line to minimise the number of workstations, ensure similar idle times at different workstations or remove hazardous parts/required components at the earliest moment possible [6].

Manual disassembly is becoming economically unattractive. To reduce labour costs and improve disassembly efficiency, robotic disassembly [7] has been considered. A robot is able to accomplish many kinds of tasks but usually requires programming with the assumption of product invariance [8]. For a product with many unforeseen EoL conditions, implementing an automated disassembly system is very challenging.

From the perspective of process automation, there is well-established research into automatically generating component relations (AND/OR graphs, interference matrices, and disassembly precedence graphs, etc.) of a product from its CAD model [9-11]. To support automatic disassembly planning, researchers have developed a number of optimisation algorithms for finding detachable components, searching for feasible disassembly sequences and generating optimal robotic motions [12-15]. Visual detection and intelligent reasoning have also been used to build a cognitive robotic system to achieve the full automation of a series of disassembly tasks [16].

However, previous research has assumed that the component relations of a product extracted from its original CAD model remain unchanged and the product condition is entirely predictable and impacts only on the processing time and the final profit [17][18]. The disassembly of an EoL product is done by a series of predetermined operations. Each operation either is executed without interruption or is able to recover from an interruption. Usually, this is true because humans are able to check the product status to determine suitable actions. However, current robotic systems cannot deal with variability. Using a predetermined order could cause incorrect and inefficient or even destructive actions.

To overcome this drawback, existing research has suggested some real-time decision strategies, sensitive analysis, and reactive approaches based on a list of predefined rules or options [19-21]. When a failure happens and cannot be corrected, dynamic disassembly re-planning is essential but rarely considered especially in robotic disassembly.

This paper proposes a detection strategy and a new intelligent optimisation algorithm to identify detachable elements in a partially disassembled product and reassign disassembly sequences and directions when irremovable components form component interlocks. This detection of subassemblies and separable components is very fast and enables a robot to adjust disassembly plans in real time. The proposed method involves first using an interference matrix [10] to record the component relations of a product and the ADD operator to find detachable components and extract further information. The "double-point" strategy is then applied to detect detachable subassemblies as interference loops. A "ternary" bees algorithm based on greedy search and meta-heuristic techniques has been developed to generate feasible disassembly re-planning solutions when the planned operation fails. The paper gives three case studies to demonstrate the efficiency of the proposed approach compared with existing heuristics and meta-heuristics.

The paper is structured as follows. Section 2 summarises the state-of-the-art in disassembly sequence planning and re-planning research. Section 3 gives a formulation of the disassembly re-planning problem and assumptions. In Section 4, the details of the double-point strategy and the complete detection process are elaborated with examples. In Section 5, we describe a local search operator and illustrate the procedure of the ternary bees algorithm with the help of the disassembly re-planning problem. We then present experimental results and an analysis of the above approach based on typical products. Concluding remarks are provided in Section 6.

2 Literature review

Focusing on the disassembly process of one specific product, a robotic disassembly plan includes sequence planning and trajectory planning [22]. Normally, the trajectory for a robot can be generated according to the disassembly sequence. Hence, the disassembly re-planning task considered in this paper mainly consists of reforming the sequence to complete the disassembly efficiently. This section briefly reviews the existing literature on the representation of EoL products, methods of automatic disassembly sequence planning, and studies on robotic disassembly re-planning.

2.1 Representation of EoL products

Product representation expresses the prerequisites of disassembly sequencing, such as the inference relations among product components, their precedence conditions, and some other restrictions. It is usually extracted from a CAD model.

Dini et al. [23] defined a product to be assembled by three matrices, i.e., interference matrix, contact matrix, and connection matrix. These matrices are then combined and widely used in disassembly in the early time. Examples are the study of Ong et al. [24] on subassembly detection, the improved description of disassembly precedence proposed by González et al. [25], and the disassembly matrix proposed by Huang et al. [26]. In addition, researchers also defined different sorts of relation matrices to adapt their disassembly models. Tao et al. [12] denoted four matrices for fasteners and components respectively to describe their feasible directions and constraints. Afsharzadeh et al. [27] introduced an adjacent matrix to guide the greedy search of disassembly sequence. Tian et al. [28] constructed a relation matrix to embody the constraint type and quantity between adjacent components. Behdad et al. [29] further included the disassembly tools and the operational directions in an expanded matrix.

Another typical expression for disassembly is AND/OR graph [30]. It is established by a group of nodes and hyper-arcs, which represent the components/subassemblies and the disassembly operations, respectively. It is more intuitive for disassembly planner to find a feasible solution [31][32]. Based on the original definition, Koc et al. [33] designed a transformed AND/OR graph (TAOG) to extensively integrate all possible disassembly path/sequence in a graph. Han et al. [34] then applied a weighted AND/OR graph to illustrate profit or environmental issues extensively. Due to that the tool-dependent operations are covered, it is non-trivial to obtain an AND/OR graph from a CAD model directly.

Currently, the interference matrix is the most commonly used representation for disassembly sequencing [35][36], while the AND/OR graph are more applied in disassembly line balancing [37][38]. Additional tree/graph representations for disassembly sequencing include the disassembly constraint graph proposed by Li et al. [39], the cooperative disassembly hierarchical tree for cooperative disassembly designed by Zhang et al. [40], the disassembly geometry contacting graph with different disassembly level presented by Mitrouchev et al. [41], and so on.

2.2 Methods of automatic disassembly sequencing

Disassembly sequence planning consists of three categories, which are, complete disassembly, partial disassembly and selective disassembly, according to the disassembly requirement. The methods for disassembly sequencing are very similar with which for assembly sequencing [5]. Ghandi et al. [42] summarised these methods and classified them as 8 kinds, graph-based method, grid-based method, sampling-based method, space decomposition method, interactive method, mathematical programming, metaheuristic, and intelligent computation method.

Recent research is more likely to design hybrid approaches based on the above techniques to deal with a different complex situation. Hui et al. [43] designed an iterative scheme to generate disassembly feasibility information graph together with a genetic algorithm (GA) for complete disassembly sequencing. Adenso-Díaz et al. [44] separated the problem into two stages and employed the greedy randomised adaptive search procedure (GRASP) and path-relinking-based heuristic to implement exploration and exploitation. Lambert et al. [45] brought the iterative nature of metaheuristic to binary integer linear programming combined with a heuristic to find a good solution. Alshibli et al. [46] introduced the tabu search mechanism to GA for disassembly sequencing. Kim et al. [47] also adopted the branch and bound algorithm with the incorporation of a shortest path heuristic to solve selective disassembly sequencing in a parallel disassembly environment.

Except that, metaheuristic is the most commonly used kind due to its high extensibility on different kinds of product models. Typical examples in recent five years are the multi-objective GA used by Rickli et al. [48] for partial disassembly sequencing, the modified teaching-learning-based optimisation algorithms designed by Xia et al. [49][50] for complete disassembly sequencing in different circumstances, the ant colony algorithm applied by Ghandi et al. [51] for general disassembly sequencing and the bees algorithm proposed by Liu et al. [35] especially for robotic disassembly sequencing.

In short, existing methods are able to provide optimal or good sub-optimal solutions for disassembly sequencing with various objectives and constraints. Nevertheless, the frequently-used graph-based methods, mathematical programming, metaheuristics, and the hybrid methods require a long time to search the solution space step by step, making them hard to adapt for re-planning in robotic disassembly.

2.3 Studies on robotic disassembly re-planning

Given a sequence plan, the disassembly process is carried out with various uncertainties as mentioned above. These uncertainties can render the original plan wholly or partly infeasible. Gungor et

al. [30] have demonstrated the possible failures and indicated the importance of robotic disassembly re-planning in their early research. Zussman et al. [20] proposed to use a pre-sorted list of transitions based on a Petri-net to adjust the disassembly when a failure happened. However, the sorted value for online decision may be inaccessible. Without the update of the sorted value and the component relations, the next transition may fail as well. Lately, ElSayed et al. [19] suggested an online decision process to generate disassembly plan for an automated robot. They introduced an optical module to the robot and detected detachable components in each iteration with a GA to determine the disassembly order of these components. However, when an operation has truly failed, there is no re-planning strategy that can be used quickly to adjust the existing operations of the robot.

3 Problem formulation

Disassembly re-planning refers to the generation of feasible sequences dynamically when one disassembly operation of the preliminary plan has failed. To avoid delays, the re-planning should be as simple and speedy as possible. According to the disassembly target, we can distinguish between re-planning algorithms for complete disassembly, partial disassembly, and selective disassembly. This paper focuses on complete disassembly in which we need only to check if the disassembly operation can continue.

3.1 Assumptions

To limit the scope of the work, we have assumed the following:

1. During each operation, only one component or subassembly is removed.
2. Each operation only involves linear movements along principal directions.
3. Tool changing is performed by humans.
4. The disassembly involves only non-destructive operations.
5. The preliminary disassembly sequence consists of both the order of component removals and their directions.
6. Each robotic operation is executed at a standard speed without extra processing.
7. When a disassembly operation fails, the corresponding component is no longer detachable. This means the system will not try and remove the specific component again.
8. There are no toxic or hazardous components.

Assumptions 1-6 apply to the case when a robotic manipulator is assigned to disassemble an EoL product with pre-determined linear movements and the manipulator requires human assistance to change tools for different disassembly tasks. Assumption 7 is made to avoid having to devise an error recovery strategy for every potential failure. Finally, as toxicity or hazardous features do not influence the efficiency of a robotic disassembly procedure, they not considered in this paper (assumption 8).

3.2 Product representation

Interference matrices, AND/OR graphs and precedence graphs are three commonly used ways to represent products in both assembly and disassembly planning. However, AND/OR graphs and precedence graphs do not contain information on how a component blocks another. This information is significant in robotic disassembly for deciding the following operation trajectory. Hence, in this paper, the interference matrix is adopted to represent the relationships between components.

Existing research has developed different kinds of interference matrices according to the disassembly requirement and prerequisites. Huang et al. [26] proposed to integrate multi-directional information into one single matrix and demonstrated the checking of detachable components using the matrix. Due to its simplicity, we use the inference matrix designed in [26] to guide the disassembly re-planning in this paper.

Specifically, an interference matrix $\boldsymbol{I}$ can be written as follows.

$$\boldsymbol{I} = \begin{bmatrix} \boldsymbol{0} & \boldsymbol{I}_{12} & \cdots & \boldsymbol{I}_{1n} \\ \boldsymbol{I}_{21} & \boldsymbol{0} & \cdots & \boldsymbol{I}_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{I}_{n1} & \boldsymbol{I}_{n2} & \cdots & \boldsymbol{0} \end{bmatrix} \tag{1}$$

where each element $\boldsymbol{I}_{ij} = [I_{ij1}, I_{ij2}, \cdots, I_{ijD}]$ is a multi-dimensional vector to represent the impact of component j on component i along D directions. n represents the total number of components in an EoL product. The directions can be either (X-, X+, Y-, Y+) in a 2-dimensional platform, or (X-, X+, Y-, Y+, Z-, Z+) for 3-dimensional operations. The value of $I_{ijd}, d \in [1, D]$ can be 0 or 1. If the component j interferes with the movement of component i in direction d, then $I_{ijd} = 1$. Otherwise, the sub-element I_{ijd} is 0. As one component will not interfere itself, the diagonal vectors of $\boldsymbol{I}$ are all set as $\boldsymbol{I}_{ii} = \boldsymbol{0} = [0, 0, \cdots, 0]$. The interference relation between two components covers both contact and indirect geometrical obstruction. Based on the interference matrix, a preliminary plan is a sequence of

components or subassemblies with their disassembly directions.

4 Dynamic disassembly re-planning

If a component cannot be disassembled, it and the components it strictly obstructs will form a subassembly. Such subassemblies can be found from the interference matrix. Whether a subassembly can be disassembled as a whole very much depends on the current situation of the product. To make an appropriate new plan, we need two steps.

- Checking whether the product can be disassembled continuously. If so, the detachable components and subassemblies are accordingly listed as a segmented rough sequence.
- Providing a plan to finish the disassembly work with the minimum disassembly time.

We will illustrate the detection strategy for detachable components or subassemblies and the dynamic re-planning algorithm in the following sections.

4.1 Detection of detachable components or subassemblies

Huang et al. [26] and Liu et al. [35] have both suggested using the Boolean OR operator to check for each component $i \in [1,n]$ if there exists one direction in which no other component obstructs its removal, i.e., $\forall j \in [1,n],\ I_{ijd} = 0$. However, this operator is deficient in the situation where only subassemblies are detachable and their components are interlocked. Take the product shown in Figure 1 as an example. Its original interference matrix with four directions, i.e., (X-, X+, Y-, Y+), is shown on the right side of the product model. The product consists of five parts (named C1 to C5), three bolts (labelled as F1, F2, and F3), and one nut (i.e., F4). All of the parts and fasteners are regarded as components here.

By using the OR operator, one can find that F1, F2, and F4 can be disassembled along Y-, X+, and Y-, respectively. If F1 and F2 have been removed and F4 fails to be disassembled due to some unknown reason, the robot will stop because none of the remaining components has mobility in any direction as indicated by the elements in the right-hand-most column being all 1.

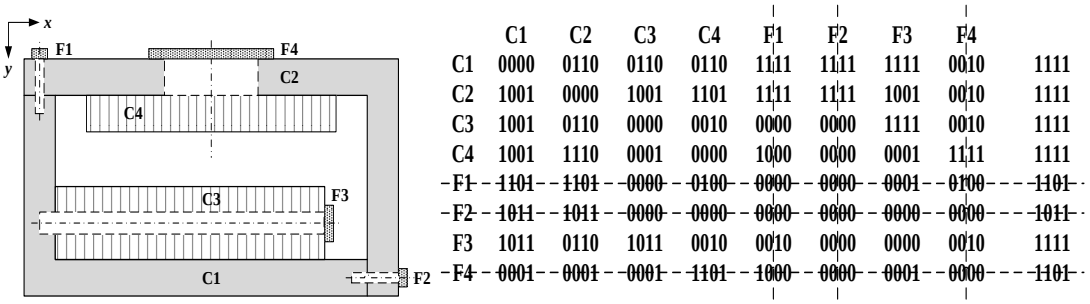


Figure 1. Example of a product to be disassembled

However, the product can be separated as two subassemblies by removing F1 and F2, no matter whether F4 is removable or not. A robotic manipulator can remove one of the subassemblies and continue to disassemble components from it separately. This "interlock" situation accompanying detachable subassemblies is more likely to take place when a failure happens in disassembly.

To solve the above problem, we replace the OR operator with the ADD operator to count the obstacles to a component in each direction and establish a double-point strategy to detect detachable subassemblies for disassembly re-planning. The pseudocode of the proposed strategy is given in Algorithm 1.

Algorithm 1. Double-point strategy for subassembly detection

Input: The interference matrix $\boldsymbol{I}$, the components which fail to be disassembled

Output: Detachable subassemblies

1 For each component $i \in [1, n]$

2 For each direction $d \in [1, D]$

3 $I_{id}^{(add)} = \sum_{j=1}^{n} I_{ijd}$

4 End For

5 $I_i^{(add)} = \min_{d=1}^{D} I_{id}^{(add)}$ and $\boldsymbol{S}_i = \varnothing$

6 For each direction $d \in [1, D]$

7 If $I_{id}^{(add)} = I_i^{(add)}$

8 $\boldsymbol{S}_i = \boldsymbol{S}_i \cup d$

9 End If

10 End For

11 End For

12 $I_{\min}^{(add)} = \arg\min_{i=1}^{n} I_i^{(add)}$

13 For each component $i \in [1, n]$

14 If $I_i^{(add)} = I_{\min}^{(add)}$ and i is not the component that failed to be disassembled

15 $\boldsymbol{V} = \boldsymbol{V} \cup i$

16 End If

17 End For

18 While $\boldsymbol{V} \notin \varnothing$

19 $v = \arg\min_{i \in \boldsymbol{V}} |\boldsymbol{S}_i|$, $\boldsymbol{V} = \boldsymbol{V} / v$

20 While $|\boldsymbol{S}_i| > 0$

21 Pop out an element d randomly from $\boldsymbol{S}_i$ and let $\boldsymbol{S}_i = \boldsymbol{S}_i / d$

22 Given an incremental list $\boldsymbol{P} = \varnothing$, $\boldsymbol{P} = \boldsymbol{P} \cup v$

23 Set the current point $k = 1$ and the reverse point $q = n$

24 While $k < |\boldsymbol{P}|$

25 For $j \in [1, n]$

26 If $I_{p_k jd} = 1$ and $j \notin \boldsymbol{P}$

27 $\boldsymbol{P} = \boldsymbol{P} \cup j$

28 Else If $I_{p_k jd} = 1$, $j \in \boldsymbol{P}$, and the position index of j in $\boldsymbol{P}$ is smaller than both k and q

29 Set q as the position of j in $\boldsymbol{P}$

30 End If

31 End For

32 If k reaches the last item of $\boldsymbol{P}$, or $|\boldsymbol{P}| = n$

33 Break

34 End If

35 $k = k + 1$

36 End While

37 If $|\boldsymbol{P}| \neq n$, p_k is not the failed component and d is not the failed direction

38 Label the items $[p_q, \cdots, p_k]$ as a subassembly, $\boldsymbol{V} = \boldsymbol{V} / (\boldsymbol{V} \cap [p_q, \cdots, p_k])$

39 Break

40 End If

41 End While

42 End While

In the above pseudocode, $I_{id}^{(add)}$ represents the result of the ADD operator for component i in direction d (Steps 2 to 4), while $\boldsymbol{S}_i$ is the direction list of component i which records the least obstructed directions (Steps 5 to 10). $I_{\min}^{(add)}$ denotes the minimal sum result for all components (Step 12). Further, a candidate list $\boldsymbol{V}$ is introduced to embody the components whose number of obstacles is equal to $I_{\min}^{(add)}$ (Steps 13 to 17).

As demonstrated by Ong et al. [24], a subassembly can be detected by checking for interference loops among the components. Steps 18 to 42 are designed to check the maximum loop between two points iteratively. To minimise the checking steps, we pick the element v which holds the shortest direction list in $\boldsymbol{V}$ as the initial element for subassembly detection (Step 19). With a randomly selected direction (Step 21), an incremental list $\boldsymbol{P}$ is defined to find the maximum interference loop (Steps 22 and 23).

In the beginning, $\boldsymbol{P}$ contains only one element $p_k = v, k = 1$. It is incremented gradually by including the obstacles to p_k one by one (Steps 26 and 27). If an obstacle has been included in $\boldsymbol{P}$ and its position in $\boldsymbol{P}$ is smaller than the current point k, this indicates that a loop exists between the current point k and the position of the obstacle. Then, the specific position is denoted as a reverse point q (Steps 29 and 30). To find the maximum loop, we need to introduce all possible obstacles to the current point k and check all of them (Steps 32 to 34). The term 'k reaches the last item of $\boldsymbol{P}$' in Step 32 means that all of the further obstacles to the current point k are included in the loop between q and k. The other condition '$|\boldsymbol{P}| = n$' indicates that all components are interlocked in this direction. If any of these conditions are satisfied, the detection stops. The maximum loop for the checked components is thus found.

Finally, if the last item of $\boldsymbol{P}$ is not the failed component and the selected d is not the failed direction, a subassembly is found. In other words, if the failed component has originally no obstacle in the d^{th} direction, k will step forward and point to the failed component. The failed component will be mistakenly

included in a subassembly without forming a loop with any previous items. To avoid this mistake, the conditions in Step 37 are adopted to label the components with qualified loops between points q and k as a subassembly and delete them from the candidate list $\boldsymbol{V}$ (Step 37 to 39).

To demonstrate the above process in detail, we take the product shown in Figure 1 as an instance. We assume that F1 and F2 have been removed and F4 is undetectable. The interference matrix with the output of the ADD operator (Steps 1 to 17) is shown in Eq. (2).

The minimal sum result is $I_{\min}^{(add)}=1$, while the candidate list is $\boldsymbol{V}=\{C1,C2,F3\}$. F3 with the shortest direction list $\boldsymbol{S}_{F3}=\{2\}$ is pushed in $\boldsymbol{P}$ and we have $d=2$. The following iteration is shown in Eq. (3). As the sum of the 2nd direction for the failed component is not 0, the 2nd component to the 4th component of $\boldsymbol{P}$, i.e., $\{C2,C4,F4\}$, form a subassembly.

$$\begin{array}{c} \\ C1 \\ C2 \\ C3 \\ C4 \\ F3 \\ F4 \end{array}\begin{bmatrix} 0000 & 0110 & 0110 & 0110 & 1111 & 0010 \\ 1001 & 0000 & 1001 & 1101 & 1001 & 0010 \\ 1001 & 0110 & 0000 & 0010 & 1111 & 0010 \\ 1001 & 1110 & 0001 & 0000 & 0001 & 1111 \\ 1011 & 0110 & 1011 & 0010 & 0000 & 0010 \\ 0001 & 0001 & 0001 & 1101 & 0001 & 0000 \end{bmatrix}\begin{array}{c} \\ 1451 \\ 4114 \\ \overset{ADD}{\Rightarrow} 2242 \\ 3224 \\ 2152 \\ 1105 \end{array} \tag{2}$$

$$\begin{aligned} &k=1,\ p_k=F3,\ \boldsymbol{P}=\{F3\}\Rightarrow\boldsymbol{P}=\{F3,C2\},\ q=n \\ &k=2,\ p_k=C2,\ \boldsymbol{P}=\{F3,C2\}\Rightarrow\boldsymbol{P}=\{F3,C2,C4\},\ q=n \\ &k=3,\ p_k=C4,\ \boldsymbol{P}=\{F3,C2,C4\}\Rightarrow\boldsymbol{P}=\{F3,C2,C4,F4\},\ q=2 \\ &k=4,\ p_k=F4,\ \boldsymbol{P}=\{F3,C2,C4,F4\}\Rightarrow\boldsymbol{P}=\{F3,C2,C4,F4\},\ q=2 \end{aligned} \tag{3}$$

After that, C1 is pushed in a new $\boldsymbol{P}$ from $\boldsymbol{V}$. Its direction list is $\boldsymbol{S}_{C1}=\{1,4\}$. We randomly choose $d=1$. The next iteration is shown in Eq. (4). The 1st component to the 3rd component of $\boldsymbol{P}$, i.e., $\{C1,F3,C3\}$, form another subassembly. The same deduction can be obtained if we choose $d=4$.

$$\begin{aligned} &k=1,\ p_k=C1,\ \boldsymbol{P}=\{C1\}\Rightarrow\boldsymbol{P}=\{C1,F3\},\ q=n \\ &k=2,\ p_k=C1,\ \boldsymbol{P}=\{C1,F3\}\Rightarrow\boldsymbol{P}=\{C1,F3,C3\},\ q=1 \\ &k=3,\ p_k=C3,\ \boldsymbol{P}=\{C1,F3,C3\}\Rightarrow\boldsymbol{P}=\{C1,F3,C3\},\ q=1 \end{aligned} \tag{4}$$

If all of the components are covered in the above subassemblies, the detection stops. The above steps are illustrated in Figure 2.

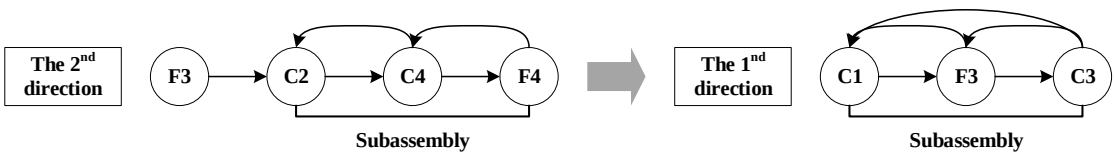


Figure 2. The process of detecting detachable subassemblies for the instance shown in Figure 1

It should be noted that the double-point strategy does not ensure finding all subassemblies at once as the EoL state of a product changes dynamically. The strategy is able to detect if a product can be further disassembled whether or not failure has occurred. If one or more subassemblies are detected, the interlock among the components is resolved. Then, further disassembly is possible. The complete procedure to detect detachable components and subassemblies can be summarised in Figure 3.

As $\boldsymbol{P}$ can hold at most n components, the time complexity of the checking process (steps 24 to 36) in the proposed strategy is $O(n)$. Assume all of the components are pushed into the candidate list and all of their direction lists hold D elements (i.e., all directions are considered). The time complexity of the double-point strategy is $O(Dn^2)$. In practice, D is no more than 6 in a 3D disassembly platform. Therefore, the time complexity of the proposed strategy is reduced to $O(n^2)$. Besides, the storage space used by $\boldsymbol{P}$, $\boldsymbol{V}$ and $\boldsymbol{S}_i, i \in [1,n]$ is $(D+2)n$ in total. So the storage space complexity of the double-point strategy is $O(n)$.

After the detection, the product instance shown in Figure 1 can be separated into two subassemblies. The interference matrix is then changed to Eq. (5). F3 is released as a detachable component, so are C1 and C3 in the next step. C2, C4 and F4 are still interlocked and cannot be further disassembled.

$$\begin{array}{c} \\ C1 \\ C2 \\ C3 \\ C4 \\ F3 \\ F4 \end{array} \begin{bmatrix} 0000 & 0\text{1}\text{1}0 & 0110 & 0\text{1}\text{1}0 & 1111 & 00\text{1}0 \\ \text{1}00\text{1} & 0000 & \text{1}00\text{1} & 1101 & \text{1}00\text{1} & 0010 \\ 1001 & 0\text{1}\text{1}0 & 0000 & 00\text{1}0 & 1111 & 00\text{1}0 \\ \text{1}00\text{1} & 1110 & 000\text{1} & 0000 & 000\text{1} & 1111 \\ 1011 & 0\text{1}\text{1}0 & 1011 & 00\text{1}0 & 0000 & 00\text{1}0 \\ 000\text{1} & 0001 & 000\text{1} & 1101 & 000\text{1} & 0000 \end{bmatrix} \begin{array}{c} \\ \\ \\ \overset{ADD}{\Rightarrow} \\ \\ \\ \end{array} \begin{array}{c} 1221 \\ 1112 \\ 2112 \\ 2221 \\ 2022 \\ 11\underline{8}2 \end{array} \tag{5}$$

A possible disassembly sequence after the failed operation on F4 is represented as shown in Eq. (6).

$$\underbrace{\{C1, C3, F3\} \circ [\mathrm{X}- \mid \mathrm{Y}+], \{C2, C4, F4\} \circ [\mathrm{X}+ \mid \mathrm{Y}-]}_{Swappable}, F3 \circ [\mathrm{X}+], \underbrace{C1 \circ [\mathrm{X}+ \mid \mathrm{Y}-], C3 \circ [\mathrm{X}- \mid Y+]}_{Swappable} \tag{6}$$

where the terms in square brackets represent possible disassembly directions for the corresponding component or subassembly. ‘Swappable’ means the order of the detachable objects can be modified to form a new sequence.

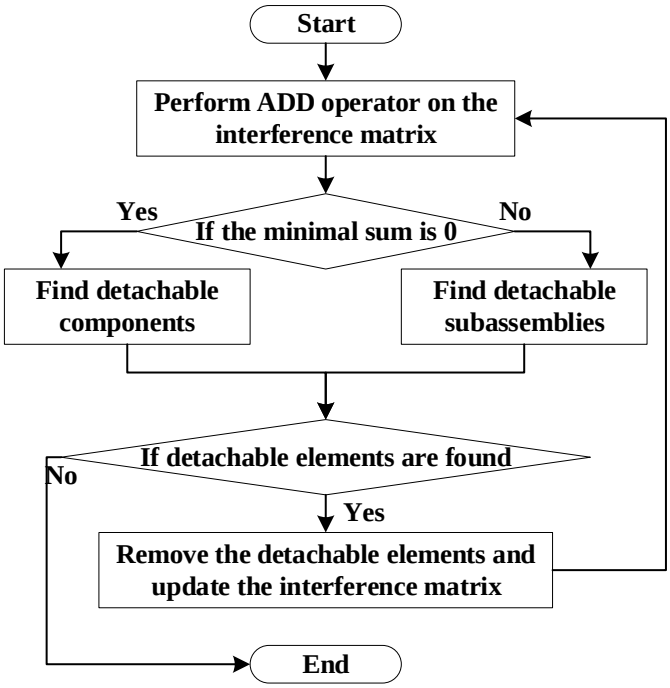


Figure 3. Procedure for detecting detachable components and subassemblies

4.2 Ternary bees algorithm for disassembly re-planning

Disassembly re-planning means re-assigning a new sequence to disassemble the remaining components and subassemblies of an EoL product. The decision variables are the same as with static disassembly sequence planning, that is, the order and direction to disassemble each component or subassembly detected from the former task, as shown in Eq. (7).

$$\boldsymbol{X} = \{x_1^{(o)}, x_2^{(o)}, \cdots, x_m^{(o)}, x_1^{(d)}, x_2^{(d)}, \cdots, x_m^{(d)}\} \tag{7}$$

where m represents the number of elements (i.e., components or subassemblies), $x_i^{(o)}, i \in [1, m]$ refers to the i^{th} element to be disassembled. It will be removed from direction $x_i^{(d)}$.

We take the total disassembly time as the main objective of disassembly re-planning. It includes not only the disassembly time for each detachable components and subassemblies, but also the time spent for moving the robotic manipulator from one position to another. The displacement time between two detachable components or subassemblies is determined by their disassembly directions and the product structure. Let the number of elements (i.e., components or subassemblies) to be disassembled as m, the basic disassembly time for each element $i \in [1, m]$ as $t_{x_i^{(o)}}$, the moving time between the gripping points of two elements $i, j \in [1, m]$ as $t^{(f)}_{x_i^{(o)}, x_j^{(o)}}$, and the direction change time between two directions

$i, j \in [1, D]$ as $t^{(d)}_{x_i^{(d)}, x_j^{(d)}}$, the total disassembly time is expressed in Eq. (8).

$$T = t_{x_i^{(o)}} + \sum_{i=0}^{m-1} t^{(f)}_{x_i^{(o)}, x_{i+1}^{(o)}} + \sum_{i=0}^{m-1} t^{(d)}_{x_i^{(d)}, x_{i+1}^{(d)}} \tag{8}$$

The sequence of disassembly re-planning is not likely to be a fully swappable sequence or a static sequence as demonstrated in [52]. It is divided into several segments according to the precedence constraints extracted from the interference matrix, as illustrated in Eq. (6). The element (which can be either a component or a subassembly) is swappable only in a specific segment.

On the one hand, existing metaheuristics for sequence planning can be too time-consuming or costly for the online re-planning as most of them integrate internal local search, external evolutionary operation, feasibility check, and solution encoding/decoding procedure step by step to update a group of individuals. On the other hand, most of the ad-hoc greedy heuristics introduce a sorting procedure to maintain a priority list and hence ignore many better solutions. Consequently, their performances are much worse than the metaheuristics, even though they can be faster. In addition, almost none of the above methods takes both the subassemblies and the disassembly directions into consideration.

Therefore, we present a ternary bees algorithm to combine the virtues of both metaheuristics and greedy heuristics to enable efficient re-planning with high solution quality. The algorithm is outlined in Figure 4. The algorithm involves a population of three individuals (i.e. potential solutions) with three operators, namely, a local search operator, an evolutionary operator and a global search operator. Each individual corresponds to one operator in an iteration.

As with the original bees algorithm [56], the population is sorted according to the fitness values of the individuals. Because there are only three individuals, the sorting is reduced to two comparisons in total to find the best, the worst and the "in-between" individual. Afterwards, the best individual is assigned to the local search operator, while the in-between individual and the worst individual are assigned to the evolutionary operator and global search operator, respectively. To make the process faster, each operator has been simplified to $O(n)$ time complexity as explained below.

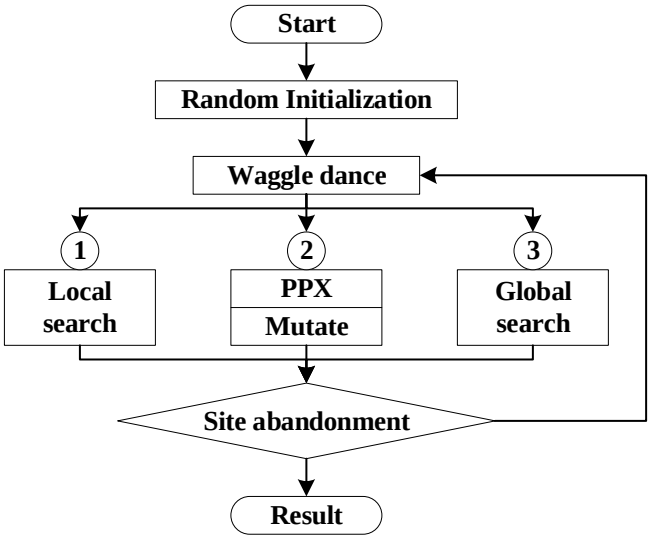


Figure 4. The mainframe of the ternary bees algorithm

(1) Local search operator

As analysed above, existing local search operators for disassembly sequence planning usually perform swap, insert, flip or inversion several times [52] with some problem-specific information. The number of iterations on local search depends on the state of the individual and its updated fitness value. To ensure only one-time search with an even use of problem-specific information, we choose a swappable position as a center from the rough sequence obtained by the component and subassembly detection procedure. Then, the nearest left or right neighbour is picked to swap with the centre by the most suitable direction change. The local search operator used in this work is expressed as Algorithm 2.

Algorithm 2. A local search operator for updating a sequence

Input: An individual $\boldsymbol{X}=\{x_1^{(o)},x_2^{(o)},\cdots,x_m^{(o)},x_1^{(d)},x_2^{(d)},\cdots,x_m^{(d)}\}$, a list $\boldsymbol{b}=\{b_0,b_1,b_2,\cdots,b_B\}$ that stores the segment points to make sure feasible swap in a sequence.

Output: The individual $\boldsymbol{X}$ updated by this proces

1 Choose a segment $(b_k,b_{k+1}]$, $0\le k<B$ which satisfies $b_{k+1}-b_k>1$

2 Select a center $b_k\le c<b_{k+1}$ and generate a random number $0<r<1$

3 If $r<0.5$ and $c>b_k+1$

4 For each position i in $(b_k,b_{k+1}]$

5 If $i\ne c$

6 Find the disassembly direction $d_{x_i^{(o)},\min}$ of $x_i^{(o)}$ from its direction list $\boldsymbol{S}_i$ that minimises the

direction change time to $x_c^{(o)}$, i.e., $d_{x_i^{(o)},\min} = \arg\min_{d \in \boldsymbol{S}_{x_i^{(o)}}} t^{(d)}_{d,x_c^{(d)}}$

Calculate the estimated moving time from $x_i^{(o)}$ to $x_c^{(o)}$, i.e., $\hat{t}_{x_i^{(o)},x_c^{(o)}} = t^{(f)}_{x_i^{(o)}x_c^{(o)}} + t^{(d)}_{d_{x_i^{(o)},\min},x_c^{(d)}}$

End if

End For

$p_l = \arg\min_{i \in [b_k, b_{k+1}), i \neq c} \hat{t}_{x_i^{(o)},x_c^{(o)}}$

$x_{p_l}^{(d)} = x_{c-1}^{(d)}$, $x_{c-1}^{(d)} = d_{x_i^{(o)},\min}$

Swap $x_{p_l}^{(o)}$ with $x_{c-1}^{(o)}$

Else

For each position i in $(b_k, b_{k+1}]$

If $i \neq c$

Find the disassembly direction $d_{x_i^{(o)},\min}$ of $x_i^{(o)}$ from its direction list $\boldsymbol{S}_i$ that minimises the direction change time from $x_c^{(o)}$, i.e., $d_{x_i^{(o)},\min} = \arg\min_{d \in \boldsymbol{S}_{x_i^{(o)}}} t^{(d)}_{x_c^{(d)},d}$

Calculate the estimated moving time from $x_c^{(o)}$ to $x_i^{(o)}$, i.e., $\hat{t}_{x_c^{(o)},x_i^{(o)}} = t^{(f)}_{x_c^{(o)}x_i^{(o)}} + t^{(d)}_{x_c^{(d)},d_{x_i^{(o)},\min}}$

End if

End For

$p_r = \arg\min_{i \in [b_k, b_{k+1}), i \neq c} \hat{t}_{x_c^{(o)},x_i^{(o)}}$

$x_{p_l}^{(d)} = x_{c+1}^{(d)}$, $x_{c+1}^{(d)} = d_{x_i^{(o)},\min}$

Swap $x_{p_l}^{(o)}$ with $x_{c+1}^{(o)}$

End If

In the above pseudocode, the segment point list $\boldsymbol{b}$ is obtained from the component and subassembly detection process shown in Algorithm 2, which stores the end position of each segment in a sequence. Take the sequence shown in Eq. (6) as an instance, there are three segments in total. The segment point list should be $\boldsymbol{b} = \{0, 2, 3, 5\}$. The first segment is from 1 to 2, the second is 3, and the third is from 4 to 5. In addition, we use $d_{x_i^{(o)},\min}$ to represent the nearest direction between $x_i^{(o)}$ and the center element $x_c^{(o)}$

and define $\hat{t}_{i,j}$ as the shortest moving time between two elements i and j.

To be specific, the local search operator means to apply the greedy search on a randomly selected center position of an individual. The position must be swappable in a segment whose size is larger than 1. The greedy search includes two sides, i.e., the left side search (Steps 3 to 12) and the right side search (Steps 13 to 23). Each time we find a detachable element in the specific segment that has the minimal disassembly cost with the centre element. If a better neighbour is found, we will swap the current neighbour with the better one and change their directions accordingly.

(2) Evolutionary operator and global search operator

As the rough sequence generated by the double-point detection strategy is segmented, traditional partially matched crossover (PMX) [53], global swap, insert, flip and inversion are no longer applicable. They will break the precedence constraints among the components and subassemblies. Only the leftmost rule, which is widely applied in precedence preserve crossover (PPX) [54],Teaching–Learning-Based Optimisation (TLBO) [49], and simplified swarm optimisation (SSO) [55], is able to maintain the precedence relation among segments by randomly selecting the leftmost variable from one parent and deleting the same variable exists in the other parent iteratively to form a new solution. Nevertheless, it ignores the selective direction optimisation. Hence, the mutation operator is introduced to perform the direction change as a complementary part of these precedence preserving operators in disassembly sequence planning.

For simplicity, we introduce the classical PPX accompanied with the single point mutation on direction change to perform sequence update for the medium individual in our re-planning algorithm. One of the parent individuals for the PPX is the medium individual given by the waggle dance. The other parent individual is randomly selected from either the best or the worst individuals. Moreover, we apply a random generation scheme to renew the worst individual and explore the whole solution space. On one hand, the good information obtained by local search is extended to the evolutionary process with high probability. On the other hand, new information is also introduced from the global search to balance the exploration and exploitation.

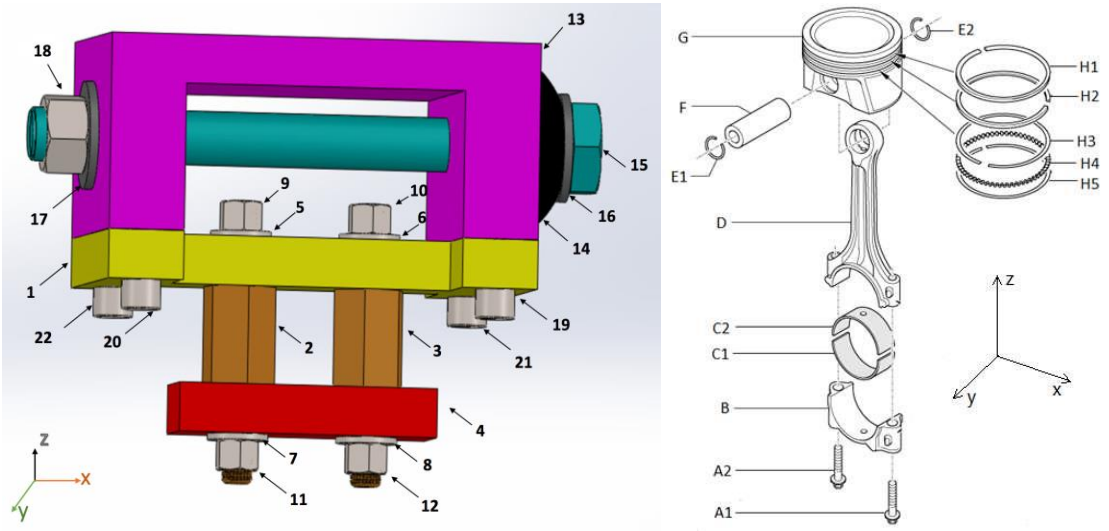
(3) Site abandonment

The site abandonment operator is a key strategy in the original bees algorithm which aims at eliminating a non-potential position in the solution space and jump out of local optimum. That is to say, if an individual is not improved for several iterations, it is no need to continue the search on that position

but to start with a new position to find diverse information. Usually, it requires a group of counters to record the evolutionary state of each individual. We assume the counter for individual i as u_i. If the individual is improved, i.e., the fitness value of the individual is reduced, $u_i = 0$. Otherwise, we perform $u_i = u_i + 1$. If the counter for an individual is larger than a threshold $u_{\max}$, it will be replace it by a newly generated sequence in accordance with the segmented precedence condition.

5 Experiments and discussions

To verify the performance of the proposed double-point detection strategy and the ternary bees algorithm, three typical products are introduced in this paper. The first model (named as Product A) is already depicted in Figure 1. We consider four directions to disassemble it. The other two products are drawn in Figure 5(a) and Figure 5(b). The second product (named as Product B) is introduced from [36], while the third one (named as Product C) is an engine piston introduced from practice. Six directions, i.e., (X-, X+, Y-, Y+, Z-, Z+) are considered to disassemble the second and the third products. The interference relationships between different components can be obtained from the product structure.



(a) Product B [36] (b) Product C

Figure 5. Two typical products applied to validate the methods proposed in this paper

Firstly, we assume three kinds of failures in disassembling each kind of product. There are 9 cases in total. In each case, the detection result obtained by the double-point strategy and its execution time are both recorded. Secondly, the ternary bees algorithm is applied to provide the re-planning solutions of the above cases. It is abbreviated as TBA and compared with two kinds of GA and two greedy schemes in

both search time and solution quality. Without loss of generality, one of the tested GAs (named as $GA^{(1)}$) consists of a tournament selection, a PPX operator, and a mutation operator, while the other (named as $GA^{(2)}$) uses a roulette wheel selection, a PPX operator, and a mutation operator as its evolutionary operators. The maximum number of iterations of the above algorithms are set as 1000 to lower the re-planning time for automatic robotic disassembly. In the greedy strategy (named as GS), we start from the leftmost position and assign the detachable element one by one by calculating the shortest moving time between the current element and the previous one. If the greedy search performs only once, we call the process as $GS^{(1)}$. Similarly, we call the greedy search as $GS^{(n)}$ if it is carried out iteratively to generate multiple solutions and provided the best. The number of iterations of $GS^{(n)}$ in this paper is set as 1000 as well.

All of the experiments are programmed by C++ on Xcode v6.1.1 platform and carried out on a PC with 2.3GHz Intel Core I7 CPU, 8GB 1600MHz DDR3 memory. As the re-planning algorithms to be tested are non-deterministic, each algorithm is run 20 times.

5.1 Performance analysis of the double-point strategy

To simplify the solution representation, we encode the components and directions for each product as shown in Table 1.

Table 1. The encoding list of the components and directions for the three products

Product A								
Component	C1	C2	C3	C4	F1	F2	F3	F4
Code	1	2	3	4	5	6	7	7
Direction	-X		+X		-Y		+Y	
Code	1		2		3		4	
Product B								
Component	No. *i*							
Code	*i*							
Direction	-X		+X		-Y		+Y	
Code	1		2		3		4	
Product C								
Component	A1	A2	B	C1	C2	D	E1	E2
Code	1	2	3	4	5	6	7	8
Direction	-X		+X		-Y		+Y	

Product A								
Component	C1	C2	C3	C4	F1	F2	F3	F4
Code	1	2	3	4	5	6	7	7
Direction	-X		+X		-Y		+Y	
Code	1		2		3		4	
Product B								
Component	No. *i*							
Code	*i*							
Direction	-X		+X		-Y		+Y	
Code	1		2		3		4	
Product C								
Component	A1	A2	B	C1	C2	D	E1	E2
Code	1	2	3	4	5	6	7	8
Direction	-X		+X		-Y		+Y	

F	G	H1	H2	H3	H4	H5
9	10	11	12	13	14	15
		-Z			+Z	
		5			6	
F	G	H1	H2	H3	H4	H5
9	10	11	12	13	14	15
		-Z			+Z	

Code	1	2	3	4	5	6

The detection results by using the process in Figure 3 for nine failure cases on different products are illustrated in Table 2. <*> defines a segmented set which contains one or more detachable elements with their possible disassembly directions. $x\circ[y]$ represents an detachable element, in which x can be a component or a subassembly and y refers to the possible directions to disassemble it. A subassembly is then embodied by a brace {*} and a direction set is further represented by a square brackets [*]. Take the first failure case of Product A as an instance, the detection result $\langle 5\circ[3],8\circ[3]\rangle,\langle\{2,6,1,3,4,7\}\circ[1,2,3,4]\rangle$ means that components 5 and 8 are disassembled in direction 3 at the first round and components 2, 6, 1, 3, 4, and 7 are left as a subassembly which can be removed only as a whole in all of the four directions.

Table 2. Detection results of the double-point strategy for the 9 cases on three kinds of EoL products

Product	No.	Non-detachable components	The detection result	Time (ms)
A	1	6	$\langle 5\circ[3],8\circ[3]\rangle,\langle\{2,6,1,3,4,7\}\circ[1,2,3,4]\rangle$	0.076
	2	8	$\langle 5\circ[3],6\circ[2]\rangle,\langle\{1,7,3\}\circ[1,4],\{2,4,8\}\circ[2,3]\rangle,\langle 7\circ[2]\rangle,\langle 1\circ[1,4],3\circ[2,3]\rangle$	0.088
	3	7,8	$\langle 5\circ[3],6\circ[2]\rangle,\langle\{1,7,3\}\circ[1,4],\{2,4,8\}\circ[2,3]\rangle$	0.080
B	4	15	$\langle 11\circ[5],12\circ[5],18\circ[0],19\circ[5],20\circ[5],21\circ[5],22\circ[5]\rangle,\langle 7\circ[5],8\circ[5],17\circ[1]\rangle,$ $\langle 4\circ[5]\rangle,\langle\{2,9,1,5,3,10,6\}\circ[3,4,5],\{16,15,13,14\}\circ[3,4,6]\rangle,\langle 9\circ[6],10\circ[6]\rangle,$ $\langle 2\circ[5],3\circ[5],5\circ[6],6\circ[6]\rangle,\langle 1\circ[1,2,3,4,5,6]\rangle$	0.314
	5	4	$\langle 11\circ[5],12\circ[5],18\circ[0],19\circ[5],20\circ[5],21\circ[5],22\circ[5]\rangle,\langle 7\circ[5],8\circ[5],17\circ[1]\rangle,$ $\langle 15\circ[2]\rangle,\langle 16\circ[2,3,4,5,6]\rangle,\langle 14\circ[2,3,4,5,6]\rangle,\langle 13\circ[3,4,6]\rangle,\langle 9\circ[6],10\circ[6]\rangle,$ $\langle 5\circ[6],6\circ[6]\rangle,\langle 1\circ[6]\rangle,\langle 2\circ[6],3\circ[6]\rangle$	0.257
	6	12,18	$\langle 11\circ[5],19\circ[5],20\circ[5],21\circ[5],22\circ[5]\rangle,\langle 7\circ[5]\rangle,$ $\langle\{9,2,1,4,5,3,6,8,10,12\}\circ[3,4,5],\{16,15,13,14,17,18\}\circ[3,4]\rangle,$ $\langle 9\circ[6],10\circ[6]\rangle,\langle 5\circ[6],6\circ[6]\rangle,\langle 1\circ[6]\rangle,\langle 2\circ[6]\rangle$	0.448
C	7	1	$\langle 2\circ[5],7\circ[4],8\circ[3],11\circ[6]\rangle,\langle 9\circ[3,4],12\circ[6]\rangle,\langle 13\circ[6]\rangle,\langle 14\circ[6]\rangle,\langle 15\circ[6]\rangle,$ $\langle 10\circ[6]\rangle,\langle\{4,3,1,6,5\}\circ[1,2,3,4,5,6]\rangle$	0.169
	8	13	$\langle 1\circ[5],2\circ[5],7\circ[4],8\circ[3],11\circ[6]\rangle,\langle 9\circ[3,4],12\circ[6]\rangle,$ $\langle\{5,6,10,13,14,15\}\circ[1,2,3,4,6],\{3,4\}\circ[1,2,3,4,5]\rangle\langle 4\circ[6],5\circ[5]\rangle,$ $\langle 3\circ[1,2,3,4,5,6],6\circ[5]\rangle$	0.18

	9	4,9	$\langle 1\circ[5],2\circ[5],7\circ[4],8\circ[3],11\circ[6]\rangle,\langle 12\circ[6]\rangle,\langle 13\circ[6]\rangle,\langle 14\circ[6]\rangle,\langle 15\circ[6]\rangle$ $\langle\{10,9,6,5\}\circ[1,2,3,4,6],\{4,3\}\circ[1,2,3,4,5]\rangle\langle 5\circ[5]\rangle$	0.184

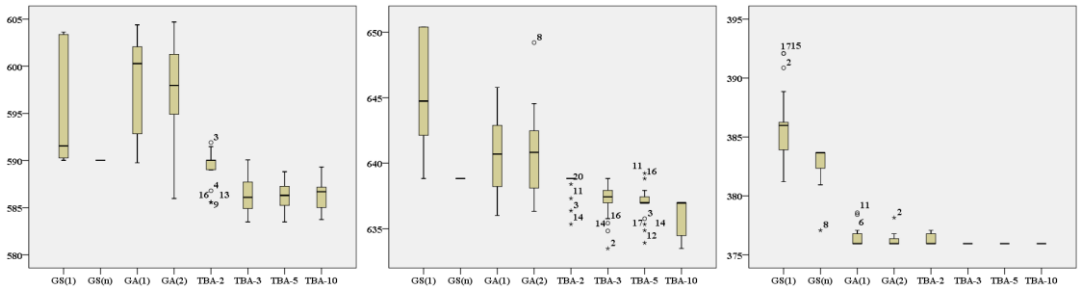
No matter given one or more components that fail to be disassembled, the proposed detection process is able to provide an accurate rough scheme to disassemble a product as complete as possible. If the non-detachable components are predictable in advance, the double-point strategy is also applicable to generate rough scheme for the static disassembly sequence planning. Similarly, if one or more failures are found in real time, the strategy will ignore the removed components and provide the segmented sets of detachable components and subassemblies for the remaining part of a product.

Normally, an EoL product contains 10 to 30 components. The execution time of the double-point strategy in the 9 cases is no more than 1ms. It depends not only on the number of components to be detected, but also on the size of subassemblies and the number of segments required to disassemble the product. As shown in Table 1, the double-point strategy takes at most 0.448ms in the 6th case, in which there are two subassemblies which contain10 components and 6 components, respectively.

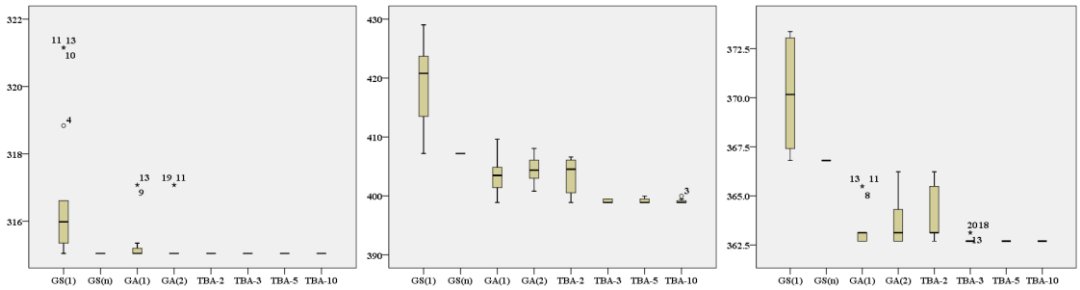
5.2 Performance analysis of the ternary bees algorithm

Because the three cases on Product A are straightforward, the best re-planning solution can be directly calculated by traversing all of the 2 to 16 solutions quickly. We apply cases 4 to 9 to test the proposed re-planning algorithm compared with the other 4 typical methods. We assume all the disassembly-related time matrices are available in advance based on the segmented component and subassembly sets obtained by the double-point detection process. For simplicity, the basic disassembly time for each element is randomly generated in the range of [10s, 30s]. The moving time between the gripping points of two elements is set in the range of [5s, 15s] as well. The direction time between two directions is also randomly produced in the range of [2s, 10s].

To accelerate the re-planning process as much as possible, the minimal population size required for the TBA should be defined. The TBA was tested with two, three, five and ten individuals and the corresponding algorithms are labeled as TBA-2, TBA-3, TBA-5 and TBA-10 in this section. For the TBA-2 that holds only 2 individuals, the better solution in each iteration is still updated by the local search operator (Algorithm 2), while the other solution is modified by selecting one of the other two operators, i.e., the global search operator and the evolutionary operator, randomly. For simplicity, the population sizes of $GS^{(n)}$, GA(1) and GA(2) are uniformly set as 10. The results of the four TBAs, the $GA^{(1)}$, the $GA^{(2)}$, the $GS^{(1)}$ and the $GS^{(n)}$ are summarised in the boxplots of Figure 6.

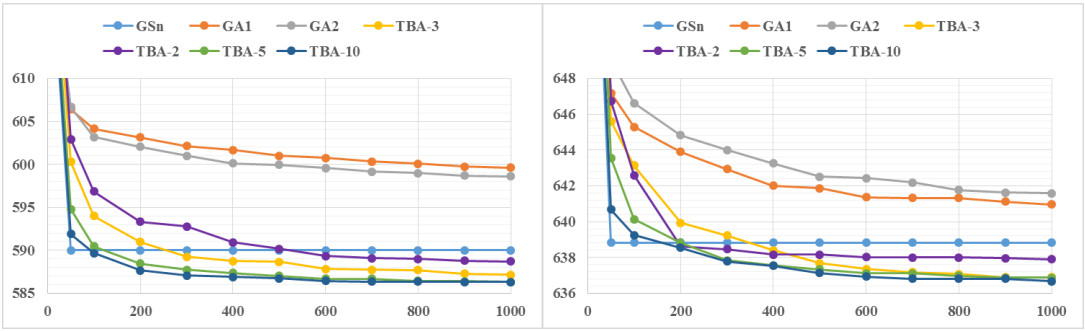


(a) case 4 (b) case 5 (c) case 6

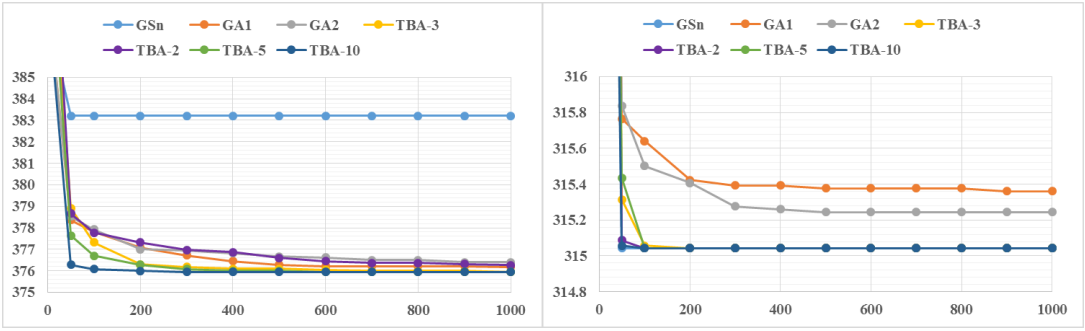


(d) case 7 (e) case 8 (f) case 9

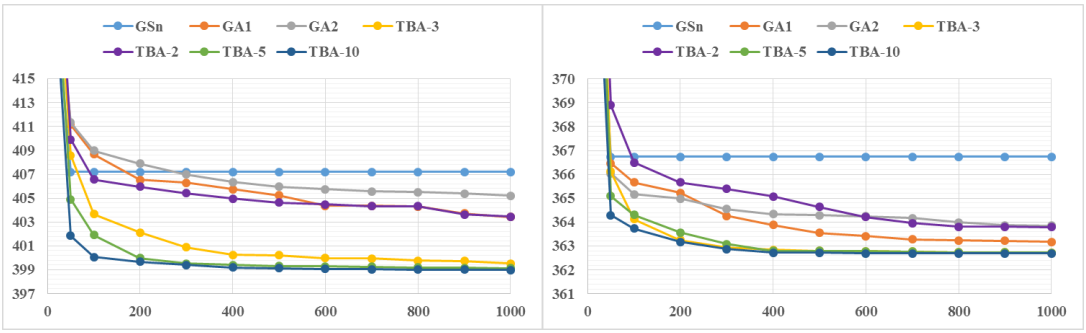
Figure 6. The fitness value of the best solution found by the 5 algorithms on six cases with respect to Product B and Product C



(a) case 4 (b) case 5



(c) case 6 (d) case 7



(e) case 8 (f) case 9

Figure 7. The average evolutionary trend of the 4 algorithms on the 6 disassembly cases with respect to Product B and Product C

By randomly selecting the start element, the $GS^{(1)}$ provided various kinds of solutions. When the number of iterations increases, the $GS^{(n)}$ quickly converged to a specific local optimum without diversity. The GAs seems to be more capable of exploration and finding some better solutions than the GSs. TBA2 performs slightly unstably with a larger solution range and more outlying points than the other three TBAs in the six cases. The performance of TBA2 is similar to that of the GAs in cases 6-9. When the population size is set as 3, the performance of the TBA is enhanced significantly. As the population size continues to increase, there is not much change in performance.

To be more specific, the average evolutionary trend of the TBAs compared with the $GS^{(n)}$ and the two GAs are demonstrated in Figure 7. Because the $GS^{(1)}$ is performed only once, there is no evolutionary trend for it. The $GS^{(n)}$ converged fast within 50 iterations and the quality of its solution fluctuates highly in different cases. The TBA-3, TBA-5 and TBA-10 have similar evolutionary trend and performs better than the $GS^{(n)}$ and the two GAs in all of the six cases. They are capable of finding better solutions in the very beginning and improving the solutions efficiently when the GAs and the TBA-2 converged gradually.

Table 3 gives the results of the pair-wise Wilcoxon test at a significant level of $\alpha = 0.05$. On the one hand, it is clear that the TBA-3 performs significantly better than the GSs and the GAs. Because both $GS^{(n)}$ and $GA^{(2)}$ can find the optimal solution quickly for case 7, $GA^{(1)}$ can find the solutions as good as TBA for case 9, the p-value between the TBA and them in these 2 cases becomes larger. Except that, the performance of the TBA is considerably better than the others, especially in the cases with more detachable elements and swappable segments, such as cases 4, 6 and 8. On the other hand, it can be seen that increasing the population size above 3 does not bring much improvement to the TBA. The results for TBA-5 and TBA-10 in all cases are similar with those for TBA-3. Hence, it is more efficient to set

the population size of the TBA as 3 to evenly perform local search, learning-based evolution and global search.

Table 3. Pair-wise Wilcoxon test between TBA and the other algorithms for 6 disassembly cases

No. of case	TBA-3 vs	$GS^{(1)}$	$GS^{(n)}$	$GA^{(1)}$	$GA^{(2)}$	TBA-2	TBA-5	TBA-10
4	*p*	0.000	0.000	0.000	0.000	0.001	0.970	0.809
5	*p*	0.000	0.000	0.001	0.001	0.000	0.530	0.059
6	*p*	0.000	0.000	0.017	0.034	0.023	1.000	1.000
7	*p*	0.000	1.000	0.038	0.083	1.000	1.000	1.000
8	*p*	0.000	0.000	0.000	0.000	0.000	0.782	0.291
9	*p*	0.000	0.000	0.005	0.008	0.000	0.083	0.083

The average search times of the 4 TBAs and the other 4 algorithms on 6 disassembly cases are shown in Table 4. Obviously, the greedy search requires the minimal number of time since it operates a single individual in each iteration. The GAs perform the worst as they need three steps of operations to select, crossover and mutate scattered individuals. These individuals may search the same solution space repeatedly. However, when a local optimum is reached, the simple random inheritance and exploration have a small chance of finding a better valley or peak position without prior knowledge. In contrast, the TBA is more efficient than the GAs in terms of both solution quality and search time even if they hold the same population size. The TBA with three individuals takes between approximately 84ms and 96ms to provide a re-planning solution for an EoL product with 15 to 22 components. When the population size increases, the search time grows linearly without much performance improvement. Therefore, maintaining three individuals and three concurrent operators is the best scheme for the TBA in solving the disassembly re-planning problem.

Table 4. Search time for the 5 algorithms on the 6 disassembly cases. (The unit of time is millisecond, i.e., ms)

No. of case	$GS^{(1)}$	$GS^{(n)}$	$GA^{(1)}$	$GA^{(2)}$	TBA-2	TBA-3	TBA-5	TBA-10
4	0.0646	23.5407	384.289	384.874	58.4483	93.7243	164.5457	334.5622
5	0.0667	23.1653	389.707	391.253	62.0667	96.2008	166.8177	338.8832
6	0.0593	22.1663	350.982	351.476	56.6708	86.2885	149.6311	302.7046
7	0.0505	22.7109	342.453	340.457	64.6471	89.1825	152.9254	298.8815
8	0.0557	22.6971	343.220	344.488	59.8267	84.4184	151.3572	300.4302
9	0.0533	21.1650	336.666	339.062	59.0556	86.3465	146.5742	297.2297

In summary, the ternary bees algorithm has the advantages of both the greedy search and the meta-

heuristic techniques. It maintains only three individuals to do exploitation by a local search strategy, learning by a semi-random inheritance, and exploration by a pure random routine collaboratively. Concurrent operations reduce the time-complexity of the algorithm significantly compared with other ad-hoc meta-heuristics. The time consumption of the ternary bees algorithm is acceptable for the real-time re-planning in a robotic disassembly environment. As the detection process consumes no more than 1ms, the total re-planning time for an EoL product will be less than 100ms in total.

6 Conclusion

This paper proposed a double-point detection strategy to find detachable components and subassemblies when a failure happens during robotic disassembly. First, we applied the ADD operator instead of the Boolean OR operator to calculate the number of obstacles using an interference matrix. Then, a list with pairs of points was established to detect subassemblies with the minimum detection times. Based on the detection, we presented a ternary bees algorithm to provide a re-planning solution combining the disassembly order and direction of not only the detachable components but also the removable subassemblies in a segmented sequence. The algorithm possesses the merits of both greedy search and meta-heuristics.

As the experimental analysis was carried out on three simple EoL products, future work will focus on more complex products to test the practical performance of the proposed approach. Moreover, the practical factors that influence disassembly cost and time from one component or subassembly to another are also a critical issue to be considered to enable a flexible re-planning process.

Acknowledgement

This paper was supported in part by the Engineering and Physical Sciences Research Council (EPSRC Grant No. EP/N018524/1).

References

[1] Ilgin M A, Gupta S M. Environmentally conscious manufacturing and product recovery (ECMPRO): A review of the state of the art. Journal of environmental management, 2010, 91(3): 563-591.

[2] Lambert A J D. Disassembly sequencing: a survey. International Journal of Production Research, 2003, 41(16): 3721-3759.

[3] Riggs R J, Battaïa O, Hu S J. Disassembly line balancing under high variety of end of life states using a joint precedence graph approach. Journal of Manufacturing Systems, 2015, 37: 638-648.

[4] Kim H J, Xirouchakis P. Capacitated disassembly scheduling with random demand. International Journal of Production Research, 2010, 48(23): 7177-7194.

[5] Kang J G, Xirouchakis P. Disassembly sequencing for maintenance: a survey. Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture, 2006, 220(10): 1697-1716.

[6] Özceylan E, Kalayci C B, Güngör A, Gupta, S M. Disassembly line balancing problem: a review of the state of the art and future directions. International Journal of Production Research, 2018: 1-23.

[7] Vongbunyong S, Kara S, Pagnucco M. Application of cognitive robotics in disassembly of products. CIRP Annals-Manufacturing Technology, 2013, 62(1): 31-34.

[8] Torres F, Puente S, Díaz C. Automatic cooperative disassembly robotic system: Task planner to distribute tasks among robots. Control Engineering Practice, 2009, 17(1): 112-121.

[9] De Mello L S H, Sanderson A C. AND/OR graph representation of assembly plans. IEEE Transactions on robotics and automation, 1990, 6(2): 188-199.

[10] Jin G Q, Li W D, Xia K. Disassembly matrix for liquid crystal displays televisions. Procedia CIRP, 2013, 11: 357-362.

[11] Gungor A, Gupta S M. Disassembly sequence plan generation using a branch-and-bound algorithm. International Journal of Production Research, 2001, 39(3): 481-509.

[12] Tao F, Bi L, Zuo Y, Nee A Y C. Partial/parallel disassembly sequence planning for complex products. Journal of Manufacturing Science and Engineering, 2018, 140(1): 011016.

[13] Smith S S, Chen W H. Rule-based recursive selective disassembly sequence planning for green design. Advanced Engineering Informatics, 2011, 25(1): 77-87.

[14] Tseng H E, Chang C C, Lee S C, Huang Y M. A Block-based genetic algorithm for disassembly sequence planning. Expert Systems with Applications, 2018, 96: 492-505.

[15] Friedrich C, Csiszar A, Lechler A, Verl A. Efficient Task and Path Planning for Maintenance Automation Using a Robot System. IEEE Transactions on Automation Science and Engineering, 2017.

[16] Vongbunyong S, Kara S, Pagnucco M. Learning and revision in cognitive robotics disassembly automation. Robotics and computer-integrated manufacturing, 2015, 34: 79-94.

[17] Rickli J L, Camelio J A. Partial disassembly sequencing considering acquired End-of-Life product age distributions. International Journal of Production Research, 2014, 52(24): 7496-7512.

[18] Kim H W, Park C, Lee D H. Selective disassembly sequencing with random operation times in parallel disassembly environment. International Journal of Production Research, 2018: 1-15.

[19] ElSayed A, Kongar E, Gupta S M, Sobh T. A robotic-driven disassembly sequence generator for End-of-Life electronic products. Journal of Intelligent & Robotic Systems, 2012, 68(1): 43-52.

[20] Zussman E, Zhou M C. A methodology for modeling and adaptive planning of disassembly processes. IEEE Transactions on Robotics and Automation, 1999, 15(1): 190-194.

[21] Lambert A J D. Linear programming in disassembly/clustering sequence generation. Computers & Industrial Engineering, 1999, 36(4): 723-738.

[22] Aguinaga I, Borro D, Matey L. Parallel RRT-based path planning for selective disassembly planning. The International Journal of Advanced Manufacturing Technology, 2008, 36(11-12): 1221-1233.

[23] Dini G, Santochi M. Automated sequencing and subassembly detection in assembly planning. CIRP annals, 1992, 41(1): 1-4.

[24] Ong N S, Wong Y C. Automatic subassembly detection from a product model for disassembly sequence generation. The international journal of advanced manufacturing technology, 1999, 15(6): 425-431.

[25] González B, Adenso-Díaz B. A scatter search approach to the optimum disassembly sequence problem. Computers & Operations Research, 2006, 33(6): 1776-1793.

[26] Huang Y M, Huang C T. Disassembly matrix for disassembly processes of products. International Journal of Production Research, 2002, 40(2): 255-273.

[27] Afsharzadeh A. Automatic disassembly task sequence planning of aircrafts at their End-of-Life. Ecole Polytechnique, Montreal, Canada, 2016.

[28] Tian G, Zhou M C, Chu J. A chance constrained programming approach to determine the optimal disassembly sequence. IEEE Transactions on Automation Science and Engineering, 2013, 10(4): 1004-1013.

[29] Behdad S, Thurston D. Disassembly and reassembly sequence planning tradeoffs under uncertainty for product maintenance. Journal of mechanical design, 2012, 134(4): 041011.

[30] Gungor A, Gupta S M. Disassembly sequence planning for products with defective parts in product recovery. Computers & Industrial Engineering, 1998, 35(1-2): 161-164.

[31] Dong J, Gibson P, Arndt G. Disassembly sequence generation in recycling based on parts accessibility and End-of-Life strategy. Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture, 2007, 221(6): 1079-1085.

[32] Yu B, Wu E, Chen C, Yang Y, Yao B Z, Lin Q. A general approach to optimize disassembly sequence planning based on disassembly network: A case study from automotive industry. Advances in Production Engineering & Management, 2017, 12(4): 305-320.

[33] Koc A, Sabuncuoglu I, Erel E. Two exact formulations for disassembly line balancing problems with task precedence diagram construction using an AND/OR graph. IIE Transactions, 2009, 41(10): 866-881.

[34] Han H J, Yu J M, Lee D H. Mathematical model and solution algorithms for selective disassembly sequencing with multiple target components and sequence-dependent setups. International Journal of Production Research, 2013, 51(16): 4997-5010.

[35] Liu J, Zhou Z, Pham D T, Xu W, Ji C, Liu Q. Robotic disassembly sequence planning using enhanced discrete bees algorithm in remanufacturing. International Journal of Production Research, 2017: 1-18.

[36] Kheder M, Trigui M, Aifaoui N. Optimization of disassembly sequence planning for preventive maintenance. The International Journal of Advanced Manufacturing Technology, 2017, 90(5-8): 1337-1349.

[37] Ren Y, Yu D, Zhang C, Tian G, Meng L, Zhou X. An improved gravitational search algorithm for profit-oriented partial disassembly line balancing problem. International Journal of Production Research, 2017: 1-15.

[38] Mete S, Çil Z A, Ağpak K, Özceylan E, Dolgui A. A solution approach based on beam search algorithm for disassembly line balancing problem. Journal of Manufacturing Systems, 2016, 41: 188-200.

[39] Li J R, Khoo L P, Tor S B. An object-oriented intelligent disassembly sequence planner for maintenance. Computers in Industry, 2005, 56(7): 699-718.

[40] Zhang X F, Zhang S Y. Product cooperative disassembly sequence planning based on branch-and-bound algorithm. The International Journal of Advanced Manufacturing Technology, 2010, 51(9-12): 1139-1147.

[41] Mitrouchev P, Wang C G, Lu L X, Li G Q. Selective disassembly sequence generation based on

lowest level disassembly graph method. The International Journal of Advanced Manufacturing Technology, 2015, 80(1-4): 141-159.

[42] Ghandi S, Masehian E. Review and taxonomies of assembly and disassembly path planning problems and approaches. Computer-Aided Design, 2015, 67: 58-86.

[43] Hui W, Dong X, Guanghong D. A genetic algorithm for product disassembly sequence planning. Neurocomputing, 2008, 71(13-15): 2720-2726.

[44] Adenso-Díaz B, García-Carbajal S, Gupta S M. A path-relinking approach for a bi-criteria disassembly sequencing problem. Computers & Operations Research, 2008, 35(12): 3989-3997.

[45] Lambert A J D, Gupta S M. Methods for optimum and near optimum disassembly sequencing. International Journal of Production Research, 2008, 46(11): 2845-2865.

[46] Alshibli M, El Sayed A, Kongar E, Sobh T M, Gupta S M. Disassembly sequencing using tabu search. Journal of Intelligent & Robotic Systems, 2016, 82(1): 69-79.

[47] Kim H W, Lee D H. An optimal algorithm for selective disassembly sequencing with sequence-dependent set-ups in parallel disassembly environment. International Journal of Production Research, 2017: 1-17.

[48] Rickli J L, Camelio J A. Multi-objective partial disassembly optimization based on sequence feasibility. Journal of manufacturing systems, 2013, 32(1): 281-293.

[49] Xia K, Gao L, Li W, Chao K M. Disassembly sequence planning using a simplified teaching–learning-based optimization algorithm. Advanced Engineering Informatics, 2014, 28(4): 518-527.

[50] Xia K, Gao L, Wang L, Li W, Li X, Ijomah W. Service-oriented disassembly sequence planning for electrical and electronic equipment waste. Electronic Commerce Research and Applications, 2016, 20: 59-68.

[51] Kheder M, Trigui M, Aifaoui N. Optimization of disassembly sequence planning for preventive maintenance. The International Journal of Advanced Manufacturing Technology, 2017, 90(5-8): 1337-1349.

[52] Ghandi S, Masehian E. A breakout local search (BLS) method for solving the assembly sequence planning problem. Engineering Applications of Artificial Intelligence, 2015, 39: 245-266.

[53] Gao L, Qian W, Li X, et al. Application of memetic algorithm in assembly sequence planning. The International Journal of Advanced Manufacturing Technology, 2010, 49(9-12): 1175-1184.

[54] Ren Y, Zhang C, Zhao F, Xiao H, Tian G. An asynchronous parallel disassembly planning based on

genetic algorithm. European Journal of Operational Research, 2018.

[55] Yeh W C. Simplified swarm optimization in disassembly sequencing problems with learning effects. Computers & Operations Research, 2012, 39(9): 2168-2177.

[56] Pham D T, Ghanbarzadeh A, Koç E, Otri S, Rahim S, Zaidi M. The bees algorithm-a novel tool for complex optimisation problems. Intelligent Production Machines and Systems. 2006: 454-459.