European Journal of Operational Research 269 (2018) 647–660

Contents lists available at ScienceDirect

European Journal of Operational Research

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

Production, Manufacturing and Logistics

An asynchronous parallel disassembly planning based on genetic algorithm

Yaping Ren[a,b], Chaoyong Zhang[a,b,*], Fu Zhao[c], Huajun Xiao[a,b], Guangdong Tian[a,b,d]

[a] School of Mechanical Science and Engineering, Huazhong University of Science and Technology, Wuhan, 430074 Hubei, China
[b] The State Key Lab of Digital Manufacturing Equipment and Technology, Huazhong University of Science and Technology, Wuhan, 430074 Hubei, China
[c] School of Mechanical Engineering, Division of Environmental and Ecological Engineering, Purdue University, West Lafayette, IN 47907, USA
[d] Transportation College, Jilin University, Changchun 130022, China

ARTICLE INFO

Article history:
Received 13 April 2017
Accepted 30 January 2018
Available online 21 February 2018

Keywords:
Combinatorial optimization
Disassembly sequence planning
Asynchronous parallel disassembly
Operation time-dependent
Metaheuristics

ABSTRACT

Disassembly is one of the most crucial remanufacturing activities. Disassembly sequence planning (DSP) is a combinatorial optimization problem and has been studied by many researchers. Conventional DSP techniques focus on sequential disassembly planning (SDP) in which only one manipulator is used to remove a single part or subassembly at a time such that it is inefficient when disassembling large or complex products. Recently, parallel disassembly has attracted some interest as it employs several manipulators to remove multiple components simultaneously. However, most of the work to date focuses on parallel disassembly techniques which require synchronization between manipulators, i.e., they must start their tasks simultaneously. This simplifies the modeling and analysis efforts but fails to fully realize the benefits of parallel disassembly. In this work, we propose asynchronous parallel disassembly planning (aPDP) which eliminates the synchronization requirement. In addition to precedence constraints, aPDP becomes highly operation time-dependent. To deal with this, we design an efficient encoding and decoding strategy for the disassembly process. In this paper, a metaheuristic approach, based on a genetic algorithm, is developed to solve the aPDP problem. The proposed algorithm is applied to four products which require disassembly processes of varying complexity, and the results are compared with two methods reported in literature. It is suggested that the proposed approach can identify faster disassembly processes, especially when solving large-scale problems.

1. Introduction

Disassembly plays a key role in the reuse and remanufacturing of end-of-life (EOL) products. Retrieving usable or repairable parts through disassembly preserves the geometry and function of the part, as well as the resources consumed during manufacturing of these parts. It has been argued that in comparison with other EOL product management options such as recycling, incineration, and landfilling, disassembly carries significant economic and environmental advantages (Li, Huang, Zhao, Sutherland, & Liu, 2017; Lu, Zhou, Xiao, Chang, & Tian, 2018; Ondemir & Gupta, 2014; Tang, Zhou, & Caudill, 2002b).

Disassembly sequence planning (DSP) is used to determine the best order for disassembling subassemblies, components, and parts from a product, according to some predetermined performance metrics (Tian, Zhou, & Chu, 2013). The disassembly of a product can be classified into two categories: sequential disassembly, where parts are removed one by one, and parallel disassembly, when multiple parts are removed simultaneously (Zhang et al., 2010). Consequently, DSP is categorized into sequential disassembly planning (SDP) and parallel disassembly planning (PDP). SDP is a classical problem that has been studied for decades (Tang, Zhou, Zussman, & Caudill, 2000). In SDP, only one part or subassembly is removed at a time. As a result, sequential disassembly follows a linear process and is relatively simple to model and optimize. However, because it is a linear process, sequential disassembly may lead to a long processing time, especially for large or complex products.

To address this issue, PDP has been developed. In contrast to SDP, more than one manipulator is employed to perform disassembly tasks and multiple parts can be removed at the same time. To fully utilize the potential of parallel disassembly to shorten processing time, coordination is needed between manipulators and the timing of different operations is extremely important. In pre-

* Corresponding author at: School of Mechanical Science and Engineering, Huazhong University of Science and Technology, Wuhan, 430074 Hubei, China.

E-mail addresses: renyp1@163.com (Y. Ren), zcyhust@hust.edu.cn (C. Zhang), fzhao@purdue.edu (F. Zhao), 1690113257@qq.com (H. Xiao), tiangd2013@163.com (G. Tian).

https://doi.org/10.1016/j.ejor.2018.01.055
0377-2217/© 2018 Elsevier B.V. All rights reserved.

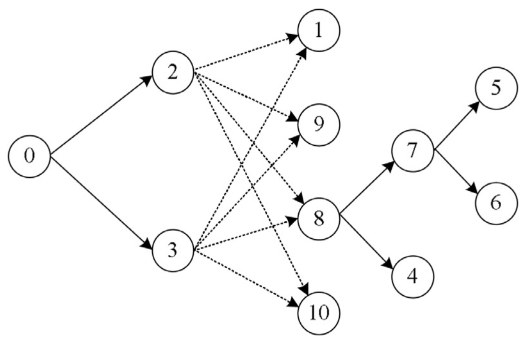

Fig. 1. The precedence diagram from McGovern and Gupta. Solid lines represent AND precedence, while dashed lines indicate OR precedence.

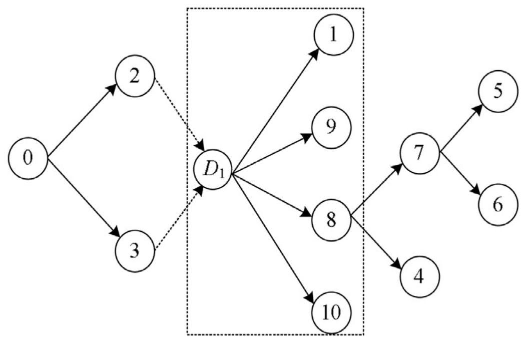

Fig. 2. A reduced precedence diagram of McGovern and Gupta example.

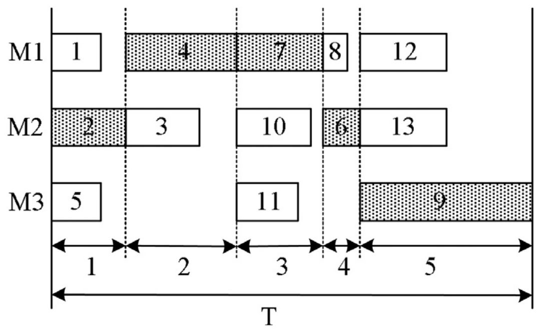

Fig. 3. A simple example of sPDP.

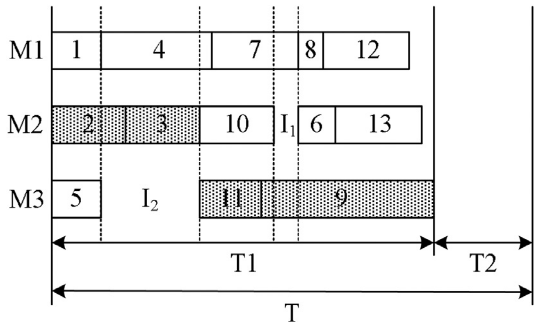

Fig. 4. The conversion from Fig. 3 showing aPDP.

vious studies, it is assumed that no manipulator can start a new task unless all other manipulators are ready for their next operation (i.e. the start time among manipulators is synchronized). This is called synchronous parallel disassembly planning (sPDP). This simplifies the problem but adds unnecessary idle time for some manipulators and may increase the total disassembly time. In this study, we propose asynchronous parallel disassembly planning (aPDP), which allows a manipulator to start its next task immediately, as long as precedence (and other) constraints are not violated.

The rest of this paper is organized as follows: Section 2 reviews the literature on disassembly planning. Section 3 describes precedence relationships in the disassembly process and presents sPDP and aPDP. Section 4 proposes an efficient metaheuristic based on a genetic algorithm (GA) to solve the aPDP problem. Section 5 shows the computational results of the proposed approach using four products with different complexities and compares our results with results obtained from methods reported in literature. Finally, Section 6 concludes our work and provides some future research suggestions.

2. Literature review

DSP can identify the best disassembly sequence/solution, among all feasible alternatives, to strip down assemblies into components without violating precedence relationships, while simultaneously maximizing economic and environmental benefits (Tang, Zhou, Zussman, & Caudill, 2002a). So far, many models and methods have been developed to solve DSP problems. Both mathematical programming and metaheuristic methods have been adopted. According to a simple disassembly graph, a linear programming model was developed to find the optimal disassembly sequence for maximum revenue (Lambert, 1999). Kang, Lee, Xirouchakis, and Persson (2001) used an integer programming model for disassembly sequencing, which was modified by applying precedence constraints to a shortest path problem. Lambert (2007) constructed an integer linear programming model for disassembly sequencing that was subject to sequence-dependent costs. In this work, an AND/OR graph was applied to graphically represent the complete set of possible disassembly operations, while an iterative method was proposed to identify the optimal solution based on a binary programming formulation.

The above papers used mathematical programming to solve DSP. Although the global optimum can be found, these methods often met challenges when the scale or the complexity of the problem increased, since DSP is a NP-hard problem (Adenso-Díaz, García-Carbajal, & Lozano, 2007; Lambert & Gupta, 2008). Some heuristic or metaheuristic methods have been developed to handle more complex problems, so that the near-optimal/optimal solution can be obtained within a reasonable time. Güngör and Gupta (2001) presented a branch-and-bound heuristic algorithm to produce disassembly sequence plans for the recycling and remanufacturing of products. A disassembly precedence matrix was defined to construct a hierarchical disassembly tree to generate feasible sequences. Another approach, which combined Petri net modeling with heuristic search procedures, was developed by Moore, Gungor, and Gupta (2001) and Rai, Rai, Tiwari, and Allada (2002). They generated disassembly sequence procedures by means of a disassembly Petri net. A heuristic was proposed for determining solutions which were ‘good enough’ for the disassembly sequencing problem; these solutions were then integrated with an exact algorithm based on an iterative method (Lambert & Gupta, 2008).

Among the metaheuristic methods, the most common approaches are intelligent algorithms, in particular, GAs. Seo, Park, and Jang (2001) adopted a disassembly AND/OR graph and developed a GA to find an optimal/near-optimal disassembly sequence. Kongar and Gupta (2006) considered three factors (i.e. disassembly demand, directions, and methods) to present a weighted multi-objective GA to solve a simple case. Hui, Dong, and Guanghong (2008) provided a disassembly feasibility information graph (DFIG) to illustrate the structure of the assembly. According to DFIG, A GA was proposed to quickly

```
Procedure: The improved GA
Input: IDP data set, GA parameters
Output: a near-optimal solution
(1)     Start
(2)     Initialize
(3)         Initialize the GA parameters: the maximum iteration count $MaxIte$, the size of population $PopSize$, the crossover and
            mutation probabilities $p_c$ and $p_m$;
(4)         Initialize population using the steps from 1 to 7 in section 4.2;
(5)     Evaluate each chromosome from the population via the decoding process;
(6)     Find the current best individual with the best fitness;
(7)     While (the termination condition has been not reached)
(8)         Selection
(9)             Perform the roulette wheel selection for the current population and determine the $PopSize$ individuals;
(10)        While (not reach the $PopSize$)
(11)            Crossover
(12)            If $p < p_c$ then // $p$ is a random number between 0 and 1
(13)                The current chromosome is regard as parent 1 and Randomly choose another one as parent 2 in
                    the population;
(14)                Cross parents 1 and 2 through the PPX;
(15)            End if;
(16)            Mutation
(17)            If $p < p_m$ then
(18)                Implement the mutation operator for the current individual;
(19)            End if;
(20)        End while;
(21)    Obtain new population and evaluate fitness;
(22)    Update the current best chromosome;
(23)    End while;
(24)    Output a near-optimal solution;
(25)    end
```

Fig. 5. Framework of the improved GA.

Table 1
Comparisons between SDP, sPDP and aPDP.

No.	Term	SDP	sPDP	aPDP
1	The precedence constraints	Yes	Yes	Yes
2	The disassembly sequence length	Yes	Yes	No
3	KPs	Yes	Yes	Yes
4	The number of parts disassembled each time	One	$\leq$ the number of manipulators	$\leq$ the number of manipulators
5	The number of the manipulator	One	More than one	More than one
6	Work areas interference	No	Yes	Yes
7	The disassembly process	Linear and not OTD	Nonlinear and partially OTD	Nonlinear and highly OTD
8	Whether the disassembly time is sequence-dependent in a complete disassembly	No	Yes	Yes
9	The operation characteristics of manipulators	No	Synchronous	Asynchronous
10	Whether the work area collision should be considered	No	Yes	Yes

search optimal solutions while minimizing disassembly costs. Tian et al. (2013) considered random variables (i.e., disassembly times and costs) to develop a chance-constrained simulation process based on a neural network and a GA. Kheder, Trigui, and Aifaoui (2015) used several parameters, such as the maintenance of usury components, the disassembly direction, the volume of the part, and the effect of changing tools, which were integrated into an objective function with different weights and were optimized simultaneously through a GA. The authors adopted a precedence preservative crossover (PPX) process which effectively ensured the feasibility of the disassembly solutions during the crossover operation of the GA. Additional intelligence algorithms, e.g., ant colony optimization (McGovern & Gupta, 2006; Shan, Li, Huang, Gao, & Li, 2007), particle swarm optimization (Zhong, Youchao, Ekene Gabriel, & Haiqiao, 2011), scatter search (González & Adenso-Díaz, 2006; Guo, Liu, Zhou, & Tian, 2016), and tabu search (Alshibli, El Sayed, Kongar, Sobh, & Gupta, 2016), have also been applied to address SDP.

All of the above studies deal with SDP, however, much less work has been done on PDP. Oliver, Chou, and Chen (1997) introduced a simplified mating graph and developed a data structure to facilitate the quick generation of a parallel disassembly sequence. Zhang and

Table 2
The initial information of predecessors for parts.

No. of parts	1	2	3	4	5	6	7	8	9	10	D_1
Predecessors	D_1	∅	∅	8	7	7	8	D_1	D_1	D_1	2 or 3
NPs	1	0	0	1	1	1	1	1	1	1	1

Table 3
The adjusted information of predecessors for parts.

No. of parts	1	2	3	4	5	6	7	8	9	10	D_1
Predecessors	D_1	∅	∅	8	7	7	8	D_1	D_1	D_1	∅
NPs	1	−1	0	1	1	1	1	1	1	1	0

Table 4
The two parents for the PPX.

No.	1	2	3	4	5	6	7	8	9	10
Parent 1	3	2	9	8	1	4	7	5	6	10
Parent 2	2	8	10	7	3	9	6	4	5	1

Table 5
The two additional strings a^1 and a^2 for the PPX.

No.	1	2	3	4	5	6	7	8	9	10
a^1	2	1	1	2	2	2	1	1	2	2
a^2	1	1	2	1	2	1	1	2	2	2

Table 6
The two offspring for the PPX.

No.	1	2	3	4	5	6	7	8	9	10
Offspring 1	2	3	9	8	10	7	1	4	6	5
Offspring 2	3	2	8	9	10	1	4	7	6	5

Table 7
Produced disassembly solutions for $M = 2$ in case study 1 ($p_c = 0.7$ and $p_m = 0.05$).

No.	The disassembly solutions	f (s)	Computational time (seconds)
1	M1: 2→ 8→ 7→ 5 M2: 3→ 10→ 9→ 1→4→ 6	89	20.14
2	M1: 3→ 1→ 9→ 7→ 5 M2: 2→ 8→ 4→ 10→ 6	90	20.02
3	M1: 2→ 8→7→ 5 M2: 3→10→ 9→ 1→ 4→ 6	89	19.96
4	M1: 2→ 10→ 9→ 1→4→ 6 M2: 3→ 8→ 7→5	91	19.57

Zhang (2010) proposed a disassembly hybrid graph model (DHGM) to explain the structural connections between parts and calculated the minimum work time of PDP via a branch-and-bound algorithm. However, the algorithm could not produce optimal/near-optimal cooperative disassembly sequences within a reasonable time for some large products. Based on a fuzzy-rough set mapping model, Zhang, Yu, Hu, Pei, and Ma (2014) used a disassembly "membership function" to group parts that could be removed at the same time. One drawback of this approach is that it can be only used to group parts into small groups of one or two parts, and it could group parts which are not physically connected. Hence, the method may produce disassembly plans that were not physically feasible. Smith and Hung (2015) used modular design theory to transform parts into modules, and generated disassembly sequences according to recursive rules. Then a GA was adopted to find optimized parallel disassembly sequences in which the feasibility of solutions can be guaranteed due to recursive rules. Unfortunately, the matrix-based product models needed to be created through manual techniques and it was complicated to use modular theory to group parts into modules. Ren et al. developed a multi-objective discrete artificial bee colony algorithm (MODABC) to minimize disassembly time and maximize profit for PDP. In this work, the proposed approach was compared with a non-dominated sorting genetic algorithm II (NSGA-II) and the results demonstrated that the MODABC had superior optimization performance (Ren, Tian, Zhao, Yu, & Zhang, 2017a). Some studies on disassembly sequencing/planning exist, which only discuss parallel operations in disassembly processes, however, these focus on SDP rather than PDP among multiple manipulators (Dutta & Woo, 1995; Kang et al., 2001; Kim & Lee, 2017).

By analyzing the existing literature, we can see that PDP has been studied less than SDP. The majority of previously published papers focus on sPDP, which requires that all cooperative disassembly tasks must begin simultaneously. sPDP can shorten the disassembly time when compared with SDP but due to synchronization among manipulators, idle time exists. In reality, it is unnecessary to require all manipulators to start a new task simultaneously, although mathematically, the synchronization greatly simplifies the modeling efforts. Therefore, removing this synchronization requirement (i.e. moving to asynchronous operation) has the potential to further shorten disassembly time and reduce associated costs.

To develop a model for aPDP, one may look into assembly literature for relevant approaches, since disassembly has been treated as an inverse assembly process (Nof & Chen, 2003). Although there are some studies on cooperative characteristics in an assembly system or in assembly sequence planning, the focus has not been, in general, on the generation and optimization of parallel assembly plans (Dong, Tong, Zhang, & Dong, 2005; Krüger, Schreck, & Surdilovic, 2011; Lee & Shin, 1990; Wang, Keshavarzmanesh, Feng, & Buchal, 2009; Zha & Lim, 2000; Zhang & Knoll, 2003).

Analysis of the aPDP problem also suggests that it shares some characteristics with job-shop scheduling problems (JSP). aPDP is highly operation time-dependent (OTD). That is, while satisfying the precedence constraints, the timing of disassembly tasks and disassembly sequences must be solved simultaneously. In JSP, jobs are assigned to machines based on their starting time and completion time. For each job, operations need to follow sequences specified in their process plan (Goncalves, Mendes, & Resende, 2005; Zhang, Li, Guan, & Rao, 2007; Zhang, Li, Rao, & Guan, 2008; Zhang, Rao, & Li, 2008). These key charactristics are similar to aPDP. The approaches used in JSP can be adapted to formulate aPDP. It is worth noting that work area collision between machines does not exist in JSP, but there may be work area collisions among manipulators when disassembling a product in PDP. Moreover, precedence constraints in aPDP are more complicated than the operation sequence in JSP.

Further, aPDP is a NP-complete problem (Zhang & Zhang, 2010), which means that, when the problem size (i.e., the number of components in the product to be disassembled) increases, the solution space is exponentially increased and it is difficult to find the optimum solution in a reasonable amount of time using an exact approach. Generally, some metaheuristics are common and effective at solving such problems in practice (Guo et al., 2016; Lambert, 2003). The GA is one of the most popular metaheuristic approaches that has been widely applied in many optimization problems (Aytug, Khouja, & Vergara, 2003; Chaudhry & Luo, 2005; Kumar, Husian, Upreti, & Gupta, 2010). As mentioned above with SDP, GAs are frequently employed in literature, however, there is only one study on sPDP (Smith & Hung, 2015). In this paper, an improved GA is developed to address the aPDP problem, which not only ensures the precedence constraints in disassembly but also takes the asynchronous operations and work area collisions into account.

3. Problem statement

To solve the aPDP problem, we first describe the mathematical and visual representations of precedence constraints during the disassembly process and the differences between the sPDP and the aPDP.

3.1. Precedence constraints

Different priorities exist among the removal of parts/components in disassembly. These precedence constraints determine the feasibility of a disassembly sequence. In other words, the precedence relationships between parts or disassembly tasks must be strictly followed when solving DSP. In general, the precedence relationships for disassembly tasks in DSP can be classified into “AND precedence”, “OR precedence”, and complex “AND/OR precedence” (Altekin, Kandiller, & Ozdemirel, 2008). Fig. 1 shows the precedence diagram of the McGovern and Gupta example (McGovern & Gupta, 2004), which will be used to in the following paragraphs to demonstrate how to formulate the problem.

In Fig. 1, each part/subassembly of a product to be disassembled is represented by a node i, where $i=1, \ldots, N$ and N is the number of disassembly tasks. Each disassembly task denotes the corresponding part i to be removed. Node 0 represents an initial point, i.e., the product to be disassembled. A part which is an AND predecessor of part i can only be disassembled before removing part i. Before disassembling part i, at least one of the parts that are OR predecessors of part i must be done. In the figure, the AND precedence task is marked by a solid line, e.g., part 0 is an AND predecessor of parts 2 and 3, while the OR precedence task is expressed via a dashed line, e.g., parts 2 and 3 are OR predecessors of parts 1, 8, 9 and 10.

To depict these relations, this work adopts precedence matrix $P=[p_{ij}]$ to represent the precedence relationship between parts i and j for a product.

$$p_{ij}=\begin{cases}1, & \text{if part } i \text{ is the AND predecessor of part } j\\ -1, & \text{if part } i \text{ is the OR predecessor of part } j\\ 0, & \text{otherwise}\end{cases}$$

Thus, precedence matrix P of the product shown in Fig. 1 is expressed as,

$$P=\begin{array}{cccccccccc|c} 1&2&3&4&5&6&7&8&9&10& \\ \hline 0&0&0&0&0&0&0&0&0&0&1\\ -1&0&0&0&0&0&0&-1&-1&-1&2\\ -1&0&0&0&0&0&0&-1&-1&-1&3\\ 0&0&0&0&0&0&0&0&0&0&4\\ 0&0&0&0&0&0&0&0&0&0&5\\ 0&0&0&0&0&0&0&0&0&0&6\\ 0&0&0&0&1&1&0&0&0&0&7\\ 0&0&0&1&0&0&1&0&0&0&8\\ 0&0&0&0&0&0&0&0&0&0&9\\ 0&0&0&0&0&0&0&0&0&0&10 \end{array}$$

Further, the precedence diagram can be simplified by defining a dummy part which is designated as Node D_k, $k=\{1, \ldots, K\}$, where K is a positive integer. Note that a dummy task does not consume time or have costs associated with it, nor does it retrieve a new part or subassembly. The precedence diagram Fig. 1 can be reduced to Fig. 2 below.

Meanwhile, precedence matrix P is adjusted as,

$$P=\begin{array}{ccccccccccc|c} 1&2&3&4&5&6&7&8&9&10&11& \\ \hline 0&0&0&0&0&0&0&0&0&0&0&1\\ 0&0&0&0&0&0&0&0&0&0&-1&2\\ 0&0&0&0&0&0&0&0&0&0&-1&3\\ 0&0&0&0&0&0&0&0&0&0&0&4\\ 0&0&0&0&0&0&0&0&0&0&0&5\\ 0&0&0&0&0&0&0&0&0&0&0&6\\ 0&0&0&0&1&1&0&0&0&0&0&7\\ 0&0&0&1&0&0&1&0&0&0&0&8\\ 0&0&0&0&0&0&0&0&0&0&0&9\\ 0&0&0&0&0&0&0&0&0&0&0&10\\ 1&0&0&0&0&0&0&1&1&1&0&11 \end{array}$$

The new P involves the dummy part D_1 listed in the last row and column, such that the size of P becomes 11×11. The reduced representation is more efficient when solving the problem, especially for a complicated product with multiple AND/OR relationships. In addition, the AND/OR relation is only considered between pairwise adjacent parts, i.e., matrix P presents local precedence constraints while keeping a successive priority among parts. For instance, part 8 precedes parts 5 and 6, and their precedence relations are guaranteed via part 7. Therefore, entries p_{85} and p_{86} are set to be 0, and p_{87}, p_{75} and p_{76} are set as 1. This expression further simplifies DSP since it is not necessary to judge the prior relationships among non-neighboring parts, especially for some large-scale products.

3.2. sPDP and aPDP

Here we use a simple example to show the difference between sPDP and aPDP. Fig. 3 shows a disassembly plan following sPDP. There are 13 parts denoted from 1 to 13, and the length of each rectangle represents the disassembly time of the corresponding part. T is the disassembly time for this disassembly process. Three manipulators (namely, M1, M2 and M3) are employed and L is the length of the disassembly sequence (which is the same as the number of synchronizations in sPDP). We can see that five synchronous operations (i.e. $L=5$) are completed and the completion time of each synchronization is determined by the maximum disassembly time among three manipulators (marked in shadow). Furthermore, M1, M2 and M3 all have some idle time, e.g., M1 and M3 are idling before the next synchronous operation begins, i.e., completing the removal of part 2 by M2.

aPDP is a more complex disassembly problem than sPDP. It eliminates the limitation in sPDP (i.e., the cooperative tasks must begin at the same time). Further, the disassembly time of each part might affect the disassembly process in aPDP. The example in Fig. 3 can be converted to aPDP as shown in Fig. 4, assuming that the conversion meets the precedence relationships. T is the disassembly time of sPDP in Fig. 3, T1 is the time consumed by aPDP, and T2 represents the time difference between aPDP and sPDP. Note that two idle intervals (I_1 and I_2) between dashed lines are required due to the precedence constraints. However, it is difficult to define the disassembly sequence length for aPDP due to its asynchronous nature. Some parts are important when determining the disassembly time, and these parts are deemed as key parts (KPs). Note that every part removed is a KP in SDP; the parts with maximum disassembly time in each synchronization are the KPs in sPDP, such as the parts marked in shadow in Fig. 3. KPs also exist in aPDP, such as the parts marked in shadow in Fig. 4.

In both sPDP and aPDP, it is necessary to consider work area collisions, which is not an issue in SDP. To the best of our knowledge, this has not been considered in previous studies. To describe this, we construct a collision matrix, $C=[c_{ij}]$, to represent when

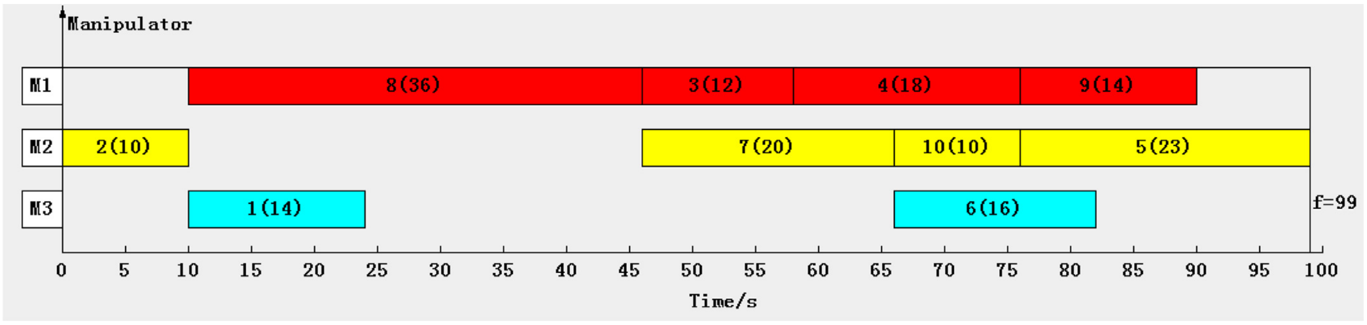

Fig. 6. The Gantt chart corresponding to the chromosome by this decoding method.

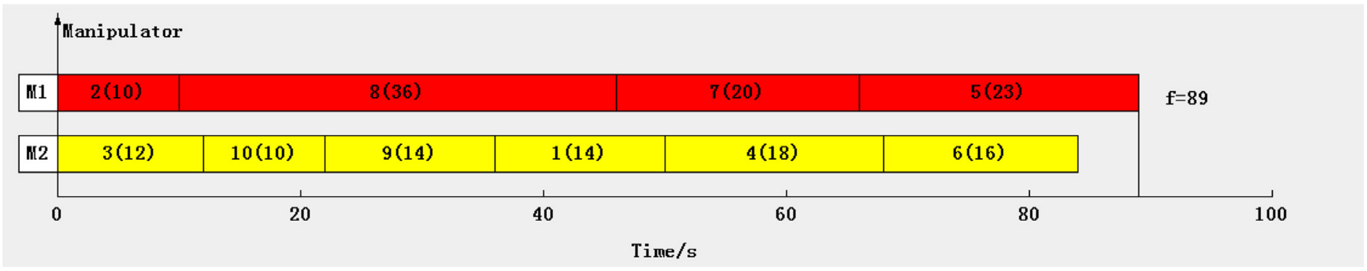

Fig. 7. The Gantt chart of the third solution in case study 1.

work areas may interfere during the disassembly process.

$$c_{ij} = \begin{cases} 1, \text{there is a work area collision between parts } i \text{ and } j \\ 0, \text{otherwise} \end{cases}$$

To show this, the example from Fig. 1 is used, where it is assumed that only parts 1 and 9 have a work area collision, so then the matrix, C, is expressed as:

$$C = \begin{array}{ccccccccccc|c} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 5 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 6 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 7 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 8 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 9 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 10 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 11 \end{array}$$

A comparison between the three disassembly planning methods are provided in Table 1. We can see that it is harder to deal with aPDP due to its highly operation time-dependence and asynchronous operations among manipulators. To simplify the problem, the current research assumes that the required resources (e.g., disassembly tools) are sufficient to disassemble the parts at the same time.

4. The proposed approach

This section presents an improved GA to solve the aPDP problem. The main steps of the proposed algorithm are shown in Fig. 5, along with pseudocode. More details are provided as follows:

4.1. Solution encoding

The encoding form of a solution directly affects the efficiency of the proposed algorithm. To effectively optimize the disassembly process, the encoding must enable the expression of a parallel disassembly sequence using more than one manipulator. Due to this encoding, solutions for SDP cannot be applied. Therefore, a double-vector list structure is adopted as the encoding form, where an individual is expressed by $v=\{v^1, v^2\}$. $v^1=\{s_1,\dots, s_i,\dots, s_N\}$ represents the disassembly parts/tasks sequence in which each element corresponds to one part or task. The elements are signified by an integer from 1 to N in which N is the number of parts in a product. $v^2=\{q_1,\dots, q_i, \dots, q_N\}$ represents the manipulator operation vector, in which each element denotes a manipulator that removes the part corresponding to v^1. q_i is encoded by a random integer between 1 and M in which M is the number of manipulators. For example, $v^1=\{2, 4, 3, 5, 8, 6, 7, 1\}$ and $v^2=\{3, 1, 1, 2, 1, 3, 2, 3\}$ show that parts 2 (s_1), 6 (s_6) and 1 (s_8) in v^1 are disassembled by manipulator 3 as seen in v^2, and so on.

The introduction of the second vector v^2 facilitates the description of a multi-manipulator cooperative process. The solution structure is easily self-adjusted with a different number of manipulators by changing the element values in v^2. Further, the alteration of v^2 cannot affect v^1, which indicates that by modifying v^2, a new solution will be created without violating the precedence constraints. This solution representation may be utilized to construct new chromosomes while avoiding the computational validation step to examine the feasibility for the new chromosomes, thus reducing the complexity of the proposed algorithm.

4.2. Population initialization

This work uses an improved GA that requires a large variety of solutions for population initialization, to obtain optimal/quasi-optimal solutions. In view of precedence relationships between subassemblies, we must ensure the feasibility of the disassembly sequence v^1 before assigning the sequence to manipulators.

From matrix P we can see the AND/OR predecessors of each task. It is relatively complicated to sequence parts only based on P. Thus, we introduce the number of predecessors (NPs) of each part to choose parts with the highest priority at that time (Ren et al., 2018). The priorities among tasks are dynamically updated as the disassembly tasks are performed. It is worth noting that NPs of some parts may be reduced as the corresponding parts are removed.

In case of the product in Fig. 2, the initial information of predecessors for all parts are listed in Table 2 according to matrix P (Ren et al., 2017b). If part 2 is disassembled first, then the data in Table 2 is adjusted as shown in Table 3. In Table 3, the precedence relationships of certain parts have been changed as the disassembly occurs. The NPs of a part equals to −1 if it is removed. A part is able to be disassembled when its NPs becomes 0, and the smaller the value, the higher the priority. Therefore, we can randomly select one part whose NPs is 0 to be disassembled until no more parts are remain. Hence, a feasible disassembly sequence v^1 is obtained for the product. According to the number of manipulators, v^2 can then be built easily by a random procedure. The generation of a feasible individual v is presented as follows:

Step 1: Start.
Step 2: Considering the precedence constraints, create matrix P, initialize NPs and, empty v^1.
Step 3: If the NPs of each part becomes −1, go to Step 6.
Step 4: Choose one part with the highest priority based on NPs and insert the selected part into the leftmost position of the vector v^1.
Step 5: Update P and NPs; Go back to Step 3.
Step 6: Randomly produce v^2 subject to the number of manipulators.
Step 7: Stop the procedure.

4.3. Decoding and solution evaluation

In this paper, we minimize the disassembly time in a parallel disassembly process that is used to assess the quality of solutions. This subsection depicts a decoding approach to calculate the disassembly time according to a given solution v.

Section 3.2 explains the concept of KPs which can directly decide on the disassembly time. But in aPDP, it is relatively complicated to decode the chromosome using KPs due to its operation time-dependence and the desynchronization of the operations. In other words, it is rather difficult to identify the KPs since each KP can be determined if and only if the total disassembly process is finalized. Fortunately, it is not necessary to find each KP. The completion of the last task means the termination of the total disassembly process, which is similar to the makespan of JSP (Karimi-Nasab & Seyedhoseini, 2013; Koulamas & Panwalkar, 2016; Pan, Tasgetiren, & Liang, 2008; Pan, Tasgetiren, Suganthan, & Chua, 2011; Zhang et al., 2007; Zhang et al., 2008). In the example proposed above, the removal of part 9 is the last operation in Fig. 4, which indicates that the completion time of part 9 equals the disassembly time. Therefore, we formulate the following objective function to minimize the disassembly time:

$$\min f = \min\left(\max_{i=1,\ldots,N} CT_i\right), \tag{1}$$

where CT_i represents the completion time of part/task i, where $i = 1, 2, \ldots N$, and N is the number of parts/tasks. The disassembly solution can be illustrated using the Gantt chart in JSP. The difference between JSP and aPDP is that the AND/OR precedence relationships and work area collisions need to be taken into account in aPDP, but do not need to be considered in JSP. Hence, we must guarantee the AND/OR priority relationships and avoid work area collisions between manipulators in order to decode successfully.

In general, the OR precedence relationship is harder to handle than the AND relationship. The OR priority relation can be removed from matrix P. Again taking the product in Fig. 2 as an example, suppose that $v^1 = \{2, 1, 8, 3, 7, 10, 4, 5, 9, 6\}$, and parts 2 and 3 are the OR predecessors of part D_1. It can be seen that part 3 is after part 2 in the sequence. Then part 3 is not deemed as the predecessor of part D_1, and part 2 is the only predecessor which implies that it becomes the AND predecessor of part D_1. Thus, matrix P is adjusted as follows (see in the solid rectangle).

$$P = \begin{array}{c} \\ \\ \end{array}\begin{array}{ccccccccccc|c}
\mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} & \mathbf{5} & \mathbf{6} & \mathbf{7} & \mathbf{8} & \mathbf{9} & \mathbf{10} & \mathbf{11} & \\
0&0&0&0&0&0&0&0&0&0&0&\mathbf{1}\\
0&0&0&0&0&0&0&0&0&0&\boxed{\mathbf{1}}&\mathbf{2}\\
0&0&0&0&0&0&0&0&0&0&\boxed{0}&\mathbf{3}\\
0&0&0&0&0&0&0&0&0&0&0&\mathbf{4}\\
0&0&0&0&0&0&0&0&0&0&0&\mathbf{5}\\
0&0&0&0&0&0&0&0&0&0&0&\mathbf{6}\\
0&0&0&0&0&0&0&0&0&0&0&\mathbf{7}\\
0&0&0&0&\mathbf{1}&\mathbf{1}&0&0&0&0&0&\mathbf{8}\\
0&0&0&\mathbf{1}&0&0&\mathbf{1}&0&0&0&0&\mathbf{9}\\
0&0&0&0&0&0&0&0&0&0&0&\mathbf{10}\\
0&0&0&0&0&0&0&0&0&0&0&\mathbf{11}
\end{array}$$

As mentioned above, the key to decoding the solution is to properly integrate the AND/OR relationships, coordination between manipulators, and work area collisions. To describe the decoding method, let DT_i denote the disassembly time of part/task i whose start time is denoted by STi. The completion time is as follows:

$$CT_i = ST_i + DT_i. \tag{2}$$

AP[i], MP[i], and CP[i] are the sets that store the AND predecessor(s) of part i, the parts immediately prior to part i for a given manipulator, and the parts that conflict with part i due to work area collisions, respectively.

The decoding procedure of an individual v is given in the following:

Step 1: Start.
Step 2: Create the aggregations of MP, AP and CP for all parts based on v, P and C, respectively.
Step 3: From left to right, assign the parts in v^1 to the corresponding manipulator in v^2 and determine their start times one by one according to MP, AP and CP. Note that the start time of part i is the maximum completion time among the parts which are included in MP[i], AP[i] and CP[i], and the completion times of all parts are computed via Eq. (2).
Step 4: Obtain the disassembly time of v by Eq. (1).
Step 5: Stop the procedure.

Using the example from Fig. 2, with a chromosome $v = \{v^1 = \{2, 1, 8, 3, 7, 10, 4, 5, 9, 6\}, v^2 = \{2, 3, 1, 1, 2, 2, 1, 2, 1, 3\}$, the disassembly time of the product is given in Table 4. Its Gantt chart is provided in Fig. 6. Note that in each rectangle, the leftmost integer represents the corresponding part and the numeric value shown in the parenthesis denotes its disassembly time.

4.4. Selection operation

The selection operation is used to choose chromosomes for reproduction in GA. The roulette wheel selection is well noted and adopted here which is a fitness-based strategy. The selection probability p_j of chromosome j is formulated as:

$$p_j = \frac{F_j}{\sum_{j=1}^{PopSize} F_j}, \tag{3}$$

where F_j and $PopSize$ represent the fitness value of chromosome j and the population size, respectively. It can be seen that the individuals with good fitness values have more "survival" opportunities, which contributes to keeping the better individuals.

4.5. Crossover operation

The crossover procedure is an important step to explore the search solution space, which directly decides the global optimization ability of the proposed method. In this problem, crossover is used to obtain better performing chromosomes by exchanging genes carried in the current good chromosomes while ensuring the feasibility of the offspring.

To do so, we utilize the precedence preservative crossover (PPX) approach to modify the disassembly sequence v^1 (Kongar & Gupta, 2001; Kheder et al., 2015). In the PPX, two additional vectors a^1 and a^2, each with the same dimensions as v^1, are randomly generated to determine the current crossing gene from one of the two parents. The elements of a^1 and a^2 are expressed by 1 or 2, respectively, where 1 and 2 represent parent 1 and parent 2, respectively. a^1 and a^2 can pass on the precedence relationships to two new offspring based on the two parent sequences while making sure that the two offspring are feasible.

An example from Fig. 2 is applied to illustrate the PPX. Tables 4 and 5 provide the two parents and additional strings, respectively. As seen in Table 5, the first element of a^1 is 2, which means that the leftmost part of parent 2 (i.e. part 2, see parent 2 in Table 4) is selected as the 1st part of offspring 1. Then part 2 is deleted in both parents such that the two parents include the rest of the 9 parts. The second one is 1 in a^1, and the leftmost of parent 1 (i.e. part 3, see parent 1 in Table 4) becomes the 2nd part of offspring 1, and so on. Finally, offspring 1 is shown in Table 6. Likewise, offspring 2 can be given according to a^2, see Table 6. The two offspring meet precedence constraints through the PPX. Furthermore, the better one is chosen to be the new chromosome between the two offspring.

4.6. Mutation operation

Since the individuals could be trapped in the local optimal solutions, mutation is employed to enhance the diversity of the population. As the crossover operation, this procedure should consider the precedence relations when inducing gene mutation for a chromosome. It is possible to yield an infeasible individual by operating a disassembly sequence v^1. Fortunately, the alteration of v^2 can avoid this while finishing the local search for the chromosome, which is explained in Section 4.1. Based on v^2, a simple mutation is designed as follows: randomly select a gene from v^2 and swap it with another gene of v^2.

5. Experimental results and analysis

The proposed approach is applied to solve multiple real-life products with different scales, and the results are compared with a branch-and-bound algorithm (Zhang & Zhang, 2010) and a fuzzy-rough set mapping model method (Zhang et al., 2014). All experiments are independently executed in Matlab and run on an Intel(R) Core(TM) i5 CPU (3.20 gigahertz/8.00 gigabytes RAM) PC with a Windows 7 operating system.

5.1. Two case studies

To demonstrate the applicability and validity of the proposed method, two products with different complexity are attempted at first. The maximum iteration number *MaxIte* and population size *PopSize* are set as 500 and 100 to ensure a good convergence when running the improved GA.

The product as shown in Fig. 2 includes AND and OR precedence relations, which is adopted as the first case study. The disassembly time of 10 parts are given in Fig. 6 and the results from four repeated trials are provided in Table 7. The first column lists the number of runs, and the second column lists the disassembly solutions obtained in each run, in which M1 and M2 represent manipulators 1 and 2, respectively. The third column shows the objective function value of the corresponding solution. The last column gives the computational time to find the solutions.

From Table 7, we can see that the disassembly times of four tests take between 89 seconds and 91 seconds and each run is able to produce similar *f* values, which indicates that the algorithm performance is stable and feasible. Moreover, each computational time is less than 21 seconds with 500 maximum iterations and a population size of 100, which is acceptable in practice. The same product will consume 173 seconds in a complete SDP which is nearly twice as much as the proposed approach. As an example, the Gantt chart of the third solution is illustrated in Fig. 7.

To further test the performance of the GA, a larger-sized product, i.e., a HG5-20 triaxial five speed mechanical transmission with 40 parts (12 components and 28 fasteners), is considered. Fig. 8 illustrates its assembly and components, and the associated matrices *P* and *C* are presented in Figs. A1 and A2 of **Appendix**, respectively. The disassembly time of part *i* (t_i) is given in Table 8, $i = 1, 2, \ldots, N$. The experiments are done as in case 1 with the same algorithm parameters (500 Iterations, population of 100 chromosomes, p_m and p_c) and the results are shown in Table 9. It should be noted that three manipulators are employed for the product, i.e., $M = 3$.

The results obtained for the larger product (Table 9) have similar characteristics as case 1. Although the *f* values of the four tests are different from each other, there is no significant difference between them. Furthermore, the best *f* value is 358 seconds, which is much smaller than the 695 seconds of processing time required by the SDP method. In summary, the proposed approach clearly outperforms SDP on the disassembly time, which is more suitable for the industrial practice. The Gantt chart of the third solution is shown in Fig. 9. With respect to the computational time, case 2 takes more time than case 1, which indicates that the computational speed decreases with increasing problem size.

5.2. Comparisons with other two methods

The two cases above compare the proposed algorithm with SDP, this subsection provides a comparison between the proposed method and two methods used to solve the sPDP, i.e., the branch-and-bound algorithm (Zhang & Zhang, 2010) and the fuzzy-rough set mapping model method (Zhang et al., 2014). The results are given in Table 10. The work area collision is not considered here to be consistent with the two studies in literature. Note that the disassembly time (730 seconds) of the last removal part is not considered in the fuzzy-rough set mapping model method. To be consistent between the three approaches, the removal time of the last part is included in the fuzzy-rough set mapping model method. Thus, the total disassembly time is 2271 seconds using the fuzzy-rough set mapping model method from the literature. The parameters of the GA are the same as the second case above. In addition, the proposed GA is implemented 10 times for each instance in order to have sufficient statistical data for comparison since the algorithm has probabilistic and randomized features. The best *f* value of each instance is given in Table 10.

It should be stressed that the two methods proposed by Zhang et al. have not considered OR precedence relations in the disassembly processes, which implies that the two products (i.e. a valve cover head fixture and an engine block in Table 10) only involve AND precedence relations. From Table 10, we can see that the disassembly times of the proposed algorithm are 20.5 seconds and 1731 seconds for the valve cover and the engine block, respectively. Both results are better than that of the other two approaches (i.e., 24 seconds and 2271 seconds). The improvement becomes even more significant in the larger scale instance with 35

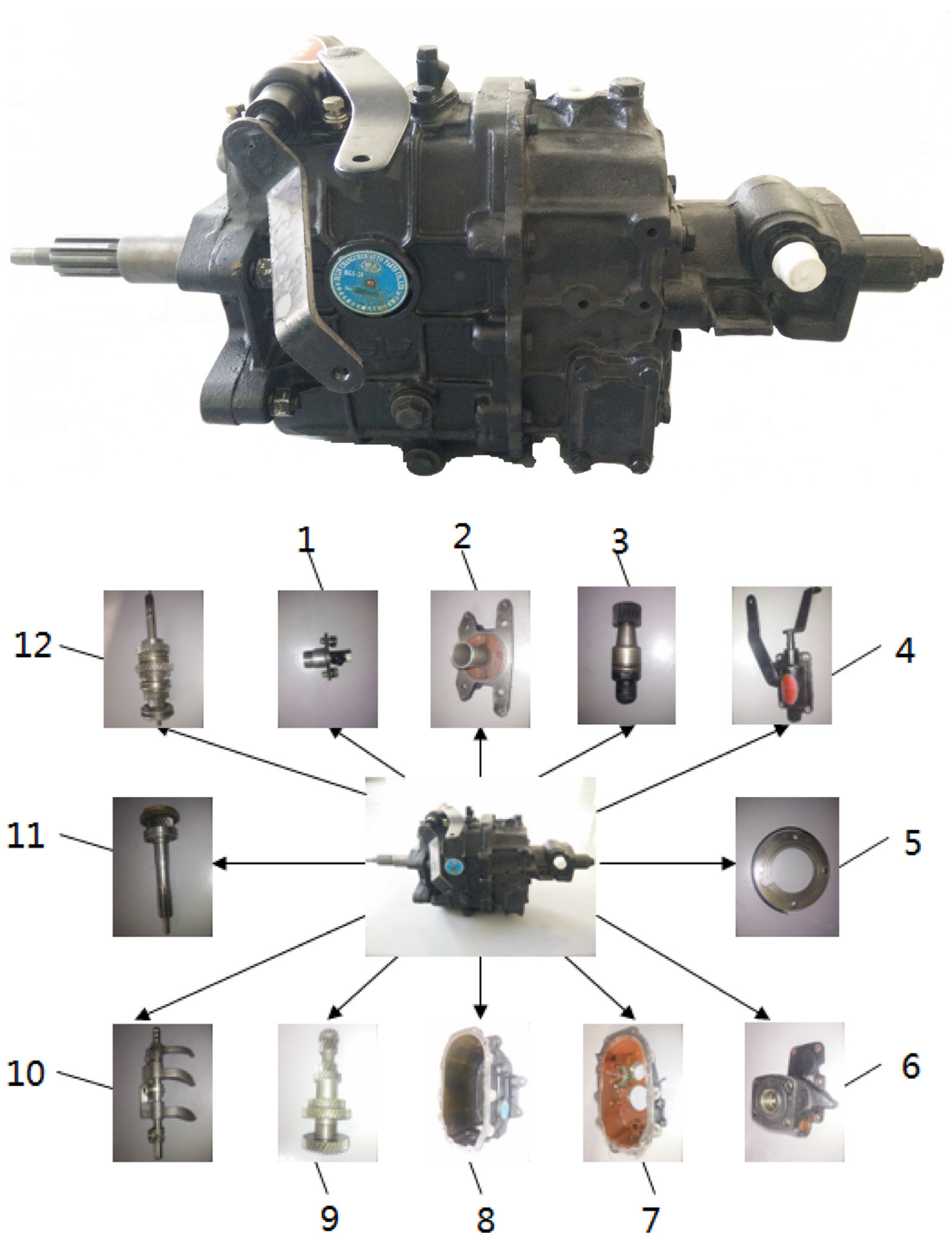

Fig. 8. The HG5-20 triaxial five speed mechanical transmission.

parts. Moreover, the behavior of the branch-and-bound algorithm is highly dependent on the product's complexity, which means that for complex/large-scale products, an inferior result may be obtained if there is a constraint on computation time. On the other hand, the fuzzy-rough set mapping model method may generate an infeasible disassembly solution (Smith & Hung, 2015). The improved GA performs better with the increase of the size and complexity of products. Since large-scale disassembly processes are more common in industry, the proposed approach is of more practical value.

5.3. Parameter sensitivity analysis

The parameters to be determined in the GA mainly include the crossover and mutation rates p_c and p_m. To identify suitable p_c and p_m, several different settings of p_c and p_m are tested using the product with 35 parts. In this subsection, the maximum number of iteration are 500, the population size is 100, and three manipulators are used. The simulation is run four times for a given value of p_c and p_m. The average disassembly time and computational time from these experiments are shown in Table 11.

We can see that the average f values are between 1731.50 seconds and 1732.25 seconds, and $p_c = 0.6$ or $p_m = 0.05$ appears to perform the worst while the difference between different parameters are not statistically significant. In addition, the best solution can be obtained when $p_c = 0.7$ and $p_m = 0.1$ or $p_c = 0.8$ and $p_m = 0.2$ as marked in bold in Table 11.

Further, the number of manipulators M is one of the most important parameters in this work, which has a great impact on the optimal disassembly solution and the objective function value. Let $MaxIte = 200$, $PopSize = 100$, $p_c = 0.7$ and $p_m = 0.1$ here. The proposed approach is executed 10 times for each product with different M. The best, average, and worst f values of the 10 runs are listed in Table 12. Note that this subsection focuses on discussing the significance of the number of manipulators M in terms of the algorithm performance, and the work area collisions are not considered.

Table 8
The value of subassemblies.

Part index	Component	Disassembly time (seconds)	Part index	Fastener	Disassembly time (seconds)
1	Inlet plug	5	13	Fastener 1	33
2	Rear bearing cover	8	14	Fastener 2	36
3	Speed sensor	10	15	Fastener 3	29
4	Transmission cover	6	16	Fastener 4	30
5	Steel rim	12	17	Fastener 5	34
6	Input shaft bearing cover	47	18	Fastener 6	36
7	Rear shell	39	19	Fastener 7	12
8	Front shell	13	20	Fastener 8	10
9	Intermediate shaft	21	21	Fastener 9	9
10	Fork	14	22	Fastener 10	16
11	Input shaft	8	23	Fastener 11	13
12	Output shaft	9	24	Fastener 12	10
			25	Fastener 13	15
			26	Fastener 14	7
			27	Fastener 15	22
			28	Fastener 16	26
			29	Fastener 17	20
			30	Fastener 18	18
			31	Fastener 19	13
			32	Fastener 20	9
			33	Fastener 21	12
			34	Fastener 22	14
			35	Fastener 23	8
			36	Fastener 24	17
			37	Fastener 25	13
			38	Fastener 26	16
			39	Fastener 27	11
			40	Fastener 28	14

Table 9
Produced disassembly solutions for $M=3$ in case study 2.

No.	The disassembly solutions	f (s)	Computational time (seconds)
1	M1: 3→19→15→22→2→26→25→23→5→31→34→35→32→7→28	361	56.56
	M2: 21→18→17→20→24→39→37→36→30→27→13→1→8→11		
	M3: 14→16→4→38→33→40→9→29→6→10→12		
2	M1: 3→18→16→2→23→33→34→31→20→28→11	358	53.34
	M2: 21→17→22→14→1→24→5→36→38→32→4→27→30→8		
	M3: 15→13→25→19→26→37→35→39→40→7→9→29→6→10→12		
3	M1: 13→17→1→25→26→32→40→31→7→9→28→10→11	364	59.89
	M2: 3→18→15→2→23→37→34→33→39→20→30→27		
	M3: 14→21→16→22→24→5→19→36→38→35→4→29→6→8→12		
4	M1: 20→17→19→13→26→35→32→34→39→22→29→30	365	57.13
	M2: 16→3→15→2→5→36→37→40→4→28→6→10→8→11		
	M3: 14→18→25→1→24→21→23→38→33→31→7→9→27→12		

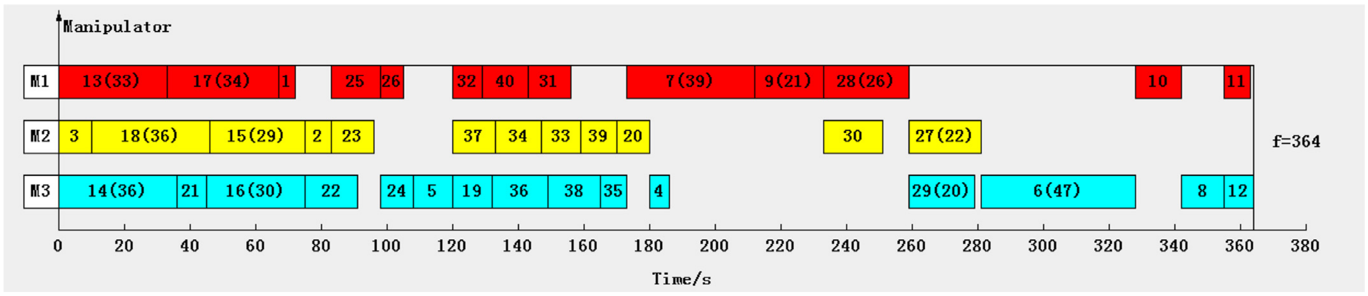

Fig. 9. The Gantt chart of the third solution in Table 9.

It can be seen that the best, average, and worst f values, which are shown as bold in Table 12, are the same when $M=1$ in each instance, since that scenario corresponds to sequential disassembly. Furthermore, for each product the best, average, and worst f values are clearly monotonically decreasing with the increase of M, while the decreasing speed seems to slow down. Especially for products 1, 2 and 3, their disassembly times finally appear to be a constant value since the best, average, and worst f values are identical at that time, which indicates that it may be not necessary to employ additional manipulators. It should be stressed that the value of M is significant for industrial practices since it is greatly related to disassembly cost and disassembly efficiency.

6. Conclusions

This work proposes a model for asynchronous parallel disassembly planning (i.e. aPDP) problem and an efficient metaheuristic based on GA is adapted to identify the disassembly sequence and manipulator allocation which minimizes the total

Table 10
Results comparison between the GA and the other two methods.

Method	Product	The number of parts	The number of manipulators	The best f (s)
The proposed GA	A valve cover head fixture taken from the literature (Zhang & Zhang, 2010)	22	2	20.5
	An engine block taken from the literature (Zhang et al., 2014)	35		1731
The-branch and-bound algorithm	A valve cover head fixture taken from the literature (Zhang & Zhang, 2010)	22		24
The fuzzy-rough set mapping model method	An engine block taken from the literature (Zhang et al., 2014)	35		2271

Table 11
Influence of p_c and p_m for the product with 35 parts.

p_c	p_m	The average f value (s)	The average computational time (seconds)
0.6	0.05	1732.00	51.13
	0.10	1732.25	50.89
	0.20	1732.00	50.96
0.7	0.05	1732.00	51.49
	0.10	**1731.50**	51.53
	0.20	1731.75	51.37
0.8	0.05	1732.00	52.56
	0.10	1731.75	52.46
	0.20	**1731.50**	52.49

Table 12
The f values with different M values for the four products.

Product id	The number of parts	The number of manipulators M	The best f (s)	The average f (s)	The worst f (s)
1	10	1	**173**	**173.0**	**173**
		2	89	90.3	93
		3	89	89	89
2	22	1	**34.5**	**34.5**	**34.5**
		2	20.5	20.4	21
		3	20	20	20
		4	18	18	18
3	35	1	**2732**	**2732**	**2732**
		2	1728	1731.9	1736
		3	1505	1516.6	1525
		4	1445	1454.9	1467
		5	1445	1445	1445
4	40	1	**695**	**695**	**695**
		2	400	414.8	426
		3	342	350.5	357
		4	306	312.8	317
		5	278	289..9	292

disassembly time. To deal with precedence relations and work area collision, a precedence matrix P and a collision matrix C are constructed and used to generate feasible disassembly sequences for aPDP. Based on the characteristics of aPDP, a Gantt chart is used to represent the asynchronous parallel disassembly process, which facilitates modeling efforts. An improved GA is developed to address this problem and was tested on four products with different scales. Results show that the proposed algorithm finds the optimal/near-optimal solution in reasonable time. Furthermore, comparison with the other two methods reported in literature, improved solutions are obtained and the degree of improvement becomes even more significant as product scale/complexity becomes larger.

Although the effectiveness and efficiency of the proposed approach have been verified, certain limitations exist. This work focuses on disassembly time only, while in reality other factors such as disassembly cost, profit, idle time may need to be taken into consideration. In addition, the proposed approach is mainly for the case when a complete disassembly is desired, but a partial or selective disassembly may represent a better solution in some scenarios. All of these need to be incorporated into future efforts on advancing disassembly planning.

Acknowledgment

This research is supported by the Funds for International Cooperation and Exchange of the National Natural Science Foundation of China (No. 51561125002), the National Natural Science Foundation of China (Grant Nos. 51575211, 51775238), the U.S. National Science Foundation (Grant No. 1512217), the 111 Project of China under Grant No. B16019, and the China Scholarship Council (Grant No. 201706160025). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Natural Science Foundation of China and the U.S. National Science Foundation. We thank Dr. Gamini Mendis at Purdue University for his feedback on the manuscript. We are grateful to the editor and anonymous referees for their constructive comments to improve this paper.

Appendix

Fig. A1, Fig. A2.

$$
P=\begin{array}{cccccccccccccccccccccccccccccccccccccccc|c}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 & 21 & 22 & 23 & 24 & 25 & 26 & 27 & 28 & 29 & 30 & 31 & 32 & 33 & 34 & 35 & 36 & 37 & 38 & 39 & 40 & \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 5 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 6 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 7 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 8 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 9 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 10 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 11 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 12 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 13 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 14 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 15 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 16 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 17 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 18 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 19 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 20 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 21 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 22 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 23 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 24 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 25 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 26 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 27 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 28 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 29 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 30 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 31 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 32 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 33 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 34 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 35 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 36 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 37 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 38 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 39 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 40
\end{array}
$$

Fig. A1. The precedence matrix of the transmission in Fig. 8.

$$C=\begin{array}{cccccccccccccccccccccccccccccccccccccccc|c}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 & 21 & 22 & 23 & 24 & 25 & 26 & 27 & 28 & 29 & 30 & 31 & 32 & 33 & 34 & 35 & 36 & 37 & 38 & 39 & 40 & \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 5 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 6 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 7 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 8 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 9 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 10 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 11 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 12 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 13 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 14 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 15 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 16 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 17 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 18 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 19 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 20 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 21 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 22 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 23 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 24 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 25 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 26 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 27 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 28 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 29 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 30 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 31 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 32 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 33 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 34 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 35 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 36 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 37 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 38 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 39 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 40
\end{array}$$

Fig. A2. The collision matrix of the transmission in Fig. 8.

References

Adenso-Díaz, B., García-Carbajal, S., & Lozano, S. (2007). An efficient GRASP algorithm for disassembly sequence planning. *OR spectrum, 29*, 535–549.

Alshibli, M., El Sayed, A., Kongar, E., Sobh, T. M., & Gupta, S. M. (2016). Disassembly sequencing using Tabu search. *Journal of Intelligent and Robotic Systems, 82*, 69–79.

Altekin, F. T., Kandiller, L., & Ozdemirel, N. E. (2008). Profit-oriented disassembly–line balancing. *International Journal of Production Research, 46*, 2675–2693.

Aytug, H., Khouja, M., & Vergara, F. E. (2003). Use of genetic algorithms to solve production and operations management problems: A review. *International Journal of Production Research, 41*, 3955–4009.

Chaudhry, S. S., & Luo, W. (2005). Application of genetic algorithms in production and operations management: A review. *International Journal of Production Research, 43*, 4083–4101.

Dong, T., Tong, R., Zhang, L., & Dong, J. (2005). A collaborative approach to assembly sequence planning. *Advanced Engineering Informatics, 19*(2), 155–168.

Dutta, D., & Woo, T. C. (1995). Algorithm for multiple disassembly and parallel assemblies. *Journal of Engineering for Industry, 117*, 102–109.

Goncalves, J. F., Mendes, J., & Resende, M. G. C. (2005). A hybrid genetic algorithm for the job shop scheduling problem. *European Journal of Operational Research, 167*(1), 77–95.

González, B., & Adenso-Díaz, B. (2006). A scatter search approach to the optimum disassembly sequence problem. *Computers & Operations Research, 33*, 1776–1793.

Güngör, A., & Gupta, S. M. (2001). Disassembly sequence plan generation using a branch-and-bound algorithm. *International Journal of Production Research, 39*, 481–509.

Guo, X., Liu, S., Zhou, M., & Tian, G. (2016). Disassembly sequence optimization for large-scale products with multiresource constraints using scatter search and petri nets. *IEEE Transactions on Cybernetics, 46*, 2435–2446.

Hui, W., Dong, X., & Guanghong, D. (2008). A genetic algorithm for product disassembly sequence planning. *Neurocomputing, 71*, 2720–2726.

Kang, J. G., Lee, D. H., Xirouchakis, P., & Persson, J. G. (2001). Parallel disassembly sequencing with sequence-dependent operation times. *CIRP Annals - Manufacturing Technology, 50*, 343–346.

Karimi-Nasab, M., & Seyedhoseini, S. M. (2013). Multi-level lot sizing and job shop scheduling with compressible process times: A cutting plane approach. *European Journal of Operational Research, 231*(3), 598–616.

Kheder, M., Trigui, M., & Aifaoui, N. (2015). Disassembly sequence planning based on a genetic algorithm. *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science, 229*, 2281–2290.

Kim, H. W., & Lee, D. H. (2017). An optimal algorithm for selective disassembly sequencing with sequence-dependent set-ups in parallel disassembly environment. *International Journal of Production Research, 55*, 7317–7333.

Kongar, E., & Gupta, S. M. (2001). Genetic algorithm for disassembly process planning. In *Proceedings of the SPIE International Conference on Environmentally Conscious Manufacturing II, Newton, Massachusetts: vol. 4569* (pp. 54–62).

Kongar, E., & Gupta, S. M. (2006). Disassembly sequencing using genetic algorithm. *The International Journal of Advanced Manufacturing Technology, 30*, 497–506.

Koulamas, C., & Panwalkar, S. S. (2016). The proportionate two-machine no-wait job shop scheduling problem. *European Journal of Operational Research, 252*(1), 131–135.

Krüger, J., Schreck, G., & Surdilovic, D. (2011). Dual arm robot for flexible and cooperative assembly. *CIRP Annals-Manufacturing Technology, 60*(1), 5–8.

Kumar, M., Husian, M., Upreti, N., & Gupta, D. (2010). Genetic algorithm: Review and application. *International Journal of Information Technology and Knowledge Management, 2*, 451–454.

Lambert, A., & Gupta, S. M. (2008). Methods for optimum and near optimum disassembly sequencing. *International Journal of Production Research, 46*, 2845–2865.

Lambert, A. J. (2007). Optimizing disassembly processes subjected to sequence-dependent cost. *Computers & Operations Research, 34*, 536–551.

Lambert, A. J. D. (1999). Linear programming in disassembly/clustering sequence generation. *Computers & Industrial Engineering, 36*, 723–738.

Lambert, A. J. D. (2003). Disassembly sequencing: A survey. *International Journal of Production Research, 41*, 3721–3759.

Lee, S., & Shin, Y. G. (1990). Assembly planning based on geometric reasoning. *Computers & Graphics, 14*(2), 237–250.

Li, L., Huang, H., Zhao, F., Sutherland, J. W., & Liu, Z. (2017). An energy-saving method by balancing the load of operations for hydraulic press. *IEEE/ASME Transactions on Mechatronics, 22*(6), 2673–2683.

Lu, Q., Zhou, G., Xiao, Z., Chang, F., & Tian, C. (2018). A selection methodology of key parts based on the characteristic of carbon emissions for low-carbon design. *International Journal of Advanced Manufacturing Technology, 94*, 3359–3373.

McGovern, S. M., & Gupta, S. M. (2004). 2-opt heuristic for the disassembly line balancing problem, *Environmentally conscious manufacturing iii: vol. 5262* (pp. 71–85). International Society for Optics and Photonics.

McGovern, S. M., & Gupta, S. M. (2006). Ant colony optimization for disassembly sequencing with multiple objectives. *The International Journal of Advanced Manufacturing Technology, 30*, 481–496.

Moore, K. E., Gungor, A., & Gupta, S. M. (2001). Petri net approach to disassembly process planning for products with complex AND/OR precedence relationships. *European Journal Of Operational Research, 135*, 428–449.

Nof, S. Y., & Chen, J. (2003). Assembly and disassembly: An overview and framework for cooperation requirement planning with conflict resolution. *Journal of Intelligent & Robotic Systems, 37*(3), 307–320.

Oliver, J. H., Chou, S. Y., & Chen, L. L. (1997). Parallel disassembly by onion peeling. *Journal of Mechanical Design, 119*, 267–274.

Ondemir, O., & Gupta, S. M. (2014). A multi-criteria decision making model for advanced repair-to-order and disassembly-to-order system. *European Journal of Operational Research, 233*, 408–419.

Pan, Q.-K., Tasgetiren, M. F., & Liang, Y.-C. (2008). A discrete particle swarm optimization algorithm for the no-wait flowshop scheduling problem. *Computers & Operations Research, 35*, 2807–2839.

Pan, Q.-K., Tasgetiren, M. F., Suganthan, P. N., & Chua, T. J. (2011). A discrete artificial bee colony algorithm for the lot-streaming flow shop scheduling problem. *Information Sciences, 181*, 2455–2468.

Rai, R., Rai, V., Tiwari, M., & Allada, V. (2002). Disassembly sequence generation: A Petri net based heuristic approach. *International Journal of Production Research, 40*, 3183–3198.

Ren, Y., Tian, G., Zhao, F., Yu, D., & Zhang, C. (2017a). Selective cooperative disassembly planning based on multi-objective discrete artificial bee colony algorithm. *Engineering Applications of Artificial Intelligence, 64*, 415–431.

Ren, Y., Zhang, C., Zhao, F., Tian, G., Lin, W., Meng, L., et al. (2018). Disassembly line balancing problem using interdependent weights-based multi-criteria decision making and 2-optimal algorithm. *Journal of Cleaner Production, 174*, 1475–1486.

Ren, Y., Yu, D., Zhang, C., Zhao, F., Tian, G., Meng, L., et al. (2017b). An improved gravitational search algorithm for profit-oriented partial disassembly line balancing problem. *International Journal of Production Research, 55*, 7302–7316.

Seo, K.-K., Park, J.-H., & Jang, D.-S. (2001). Optimal disassembly sequence using genetic algorithms considering economic and environmental aspects. *The International Journal of Advanced Manufacturing Technology, 18*, 371–380.

Shan, H., Li, S., Huang, J., Gao, Z., & Li, W. (2007). Ant colony optimization algorithm-based disassembly sequence planning. In *Proceedings of the International Conference on Mechatronics and Automation* (pp. 867–872). IEEE.

Smith, S., & Hung, P.-Y. (2015). A novel selective parallel disassembly planning method for green design. *Journal of Engineering Design, 26*, 283–301.

Tang, Y., Zhou, M., Zussman, E., & Caudill, R. (2000). Disassembly modeling, planning and application: A review. In *Proceedings of the IEEE International Conference on Robotics and Automation: vol. 3* (pp. 2197–2202). IEEE.

Tang, Y., Zhou, M., Zussman, E., & Caudill, R. (2002a). Disassembly modeling, planning, and application. *Journal of Manufacturing Systems, 21*, 200–217.

Tang, Y., Zhou, M. C., & Caudill, R. J. (2002b). An integrated approach to disassembly planning and demanufacturing operation. *IEEE Transactions on Robotics & Automation, 17*, 773–784.

Tian, G., Zhou, M., & Chu, J. (2013). A chance constrained programming approach to determine the optimal disassembly sequence. *IEEE Transactions on Automation Science and Engineering, 10*, 1004–1013.

Wang, L., Keshavarzmanesh, S., Feng, H.-Y., & Buchal, R. O. (2009). Assembly process planning and its future in collaborative manufacturing: A review. *The International Journal of Advanced Manufacturing Technology, 41*, 132.

Zha, X., & Lim, S. (2000). Assembly/disassembly task planning and simulation using expert Petri nets. *International Journal of Production Research, 38*(15), 3639–3676.

Zhang, C. Y., Li, P. G., Guan, Z. L., & Rao, Y. Q. (2007). A tabu search algorithm with a new neighborhood structure for the job shop scheduling problem. *Computers & Operations Research, 34*, 3229–3242.

Zhang, C. Y., Li, P. G., Rao, Y. Q., & Guan, Z. L. (2008). A very fast TS/SA algorithm for the job shop scheduling problem. *Computers & Operations Research, 35*, 282–294.

Zhang, C. Y., Rao, Y. Q., & Li, P. G. (2008). An effective hybrid genetic algorithm for the job shop scheduling problem. *International Journal of Advanced Manufacturing Technology, 39*(9-10), 965–974.

Zhang, J., & Knoll, A. (2003). A two-arm situated artificial communicator for human-robot cooperative assembly. *IEEE Transactions on Industrial Electronics, 50*(4), 651–658.

Zhang, X. F., Yu, G., Hu, Z. Y., Pei, C. H., & Ma, G. Q. (2014). Parallel disassembly sequence planning for complex products based on fuzzy-rough sets. *The International Journal of Advanced Manufacturing Technology, 72*, 231–239.

Zhang, X. F., & Zhang, S. Y. (2010). Product cooperative disassembly sequence planning based on branch-and-bound algorithm. *The International Journal of Advanced Manufacturing Technology, 51*, 1139–1147.

Zhong, L., Youchao, S., Ekene Gabriel, O., & Haiqiao, W. (2011). Disassembly sequence planning for maintenance based on metaheuristic method. *Aircraft Engineering and Aerospace Technology, 83*, 138–145.