Expert Systems With Applications 96 (2018) 492–505

Contents lists available at ScienceDirect

Expert Systems With Applications

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

A Block-based genetic algorithm for disassembly sequence planning

Hwai-En Tseng[a,*], Chien-Cheng Chang[b], Shih-Chen Lee[c], Yu-Ming Huang[a]

[a] Department of Industrial Engineering and Management, National Chin-Yi University of Technology, 57, Section 2, Zhongshan, Taiping District, Taichung, 41170 Taiwan, ROC
[b] Department of Industrial Design, National United University, 1, Lienda, Miaoli 36003, Taiwan, ROC
[c] Department of Innovative Living Design, Overseas Chinese University, 100, Chiao Kwang Rd., Taichung 40721, Taiwan, ROC

ARTICLE INFO

Article history:
Received 12 July 2017
Revised 31 October 2017
Accepted 1 November 2017
Available online 2 November 2017

Keywords:
Disassembly sequence planning
Block-based genetic algorithms
Kongar and Gupta's genetic algorithms
Dijkstra's algorithms

ABSTRACT

Disassembly sequence planning refers to the study of the sequential order of disassembly based on the limit attributes of parts in the process of disassembly after the product design. The present study proposes a new block-based genetic algorithm for disassembly sequence planning based upon the comparison of Kongar and Gupta's genetic algorithm and Dijkstra's algorithms. It is expected that the disassembly sequence planning problem can be solved more efficiently. Four examples are used to test the algorithms developed in this study. Finally, it shown that the quality of the solution generates met the expected effect.

1. Introduction

Disassembly sequence planning (DSP) signifies the process of disassembling a product. In product life cycle assessment, the investigation of DSP will be helpful for product evaluation. In practice, a designer often works out the product disassembly sequence according to the parts' engineering relationships based on their own experience. For this reason, a scientific assessment method will be able to contribute to the objective evaluation of product design (Ali Ilgin & Gupta, 2010).

In the past, DSP research and assembly sequence planning (ASP) were often mixed together. However, ASP must take into account the relationship between parts and consider other related constraints such as assembly time, geometric characteristics, disassembly tools and machines to determine the priority sequential order in product assembly. Traditionally, ASP solves this problem by first turning the product model into a so-called liaison graph and then searching for solutions with graph theory and an exhaustive search method (Ghandi & Msaehian, 2015). Regarding this problem, Homem De Mello and Sanderson (1991) consider ASP as the reverse process of DSP. For DSP classification, the disassembly can be divided into total disassembly and selective disassembly (Srinivasan, Figueroa, & Gadh, 1999). The total disassembly mainly occurs in a task in which the target parts are clear, so the focus is to find the best objective (such as minimum cost or maximum benefit) under various constraints. On the other hand, selective disassembly focuses on the search for the parts or modules worth disassembling. At the end of the product life, recycling companies can process related tasks in accordance with the default recycling schedule. Researchers can learn the outline of the disassembly planning study from past articles or books (Ghandi & Msaehian, 2015; Lambert & Gupta, 2005). This study focuses on the exploration of total disassembly.

Though it is possible to find the feasible or optimal solutions based on graph theory or related heuristic methods, the size of a problem solutions grows exponentially with the number of components (NP-completeness). Consequently, it has been important issue in recent years for scholars to search for more efficient methods. Regarding this, one feasible method is mathematical programming (MP), such as linear Programming (Lambert, 1999), branch-and-bound (Güngör & Gupta, 2001), integer programming methods (Lambert, 2007), and shortest path algorithms (Zhong, Youchao, Gabriel, & Haiqiao, 2011). MP methods require highly abstract modeling thinking and prior precise definitions. They may not be suitable for more detailed considerations such as application in product design or disassembly line design (Lambert, 2003). Other feasible options are offered by heuristic algorithms. Some examples are the neural network model (Huang, Wang, & Johnson, 2000), ant colony optimization (McGovern & Gupta, 2006; Wang, Liu, Li, & Zhong, 2003), particle swarm optimization (Zhong et al., 2011), scatter search (González & Adenso-Díaz, 2006; Guo, Liu, Zhou, & Tian, 2016), GRASP algorithms (Adenso-Díaz, García-Carbajal, & Lozano, 2007; Andrés, Lozano, & Adenso-Díaz, 2007), Tabu search (Alshibli, El Sayed, Kongar, Sobh, & Gupta, 2016), and genetic

* Corresponding author.
E-mail addresses: hwai_en@seed.net.tw (H.-E. Tseng), changcc@nuu.edu.tw (C.-C. Chang), d821104@ocu.edu.tw (S.-C. Lee), s3a315103@student.ncut.edu.tw (Y.-M. Huang).

https://doi.org/10.1016/j.eswa.2017.11.004
0957-4174/

algorithms (Hui, Dong, & Guanghong, 2008; Kheder, Trigui, & Aifaoui, 2015; Rickli & Camelio, 2013; Seo, Park, & Jane, 2001). These pioneers have proposed many feasible methods to solve the DSP problems.

Among these methods, genetic algorithms (GAs) apply the crossover and mutation mechanism for the new solutions in the evolution process and retain better chromosomes for the next generation. In the past, Kongar and Gupta (2006) developed the so-called Precedence Preservative Crossover (PPX) mechanism; Giudice and Fargione (2007) considered the disassembly time, life cycle cost, and environmental impact as the fitness function considerations; Hui, Dong, and Guanghong (2008) explored the Disassembly Feasibility Information Graph (DFIG) mechanism; Go, Wahab, Rahman, Ramli, and Hussain (2012) adopted a mechanism similar to that of Kongar and Gupta (2006) to develop DSP GAs. All of these scholars take advantage of GAs because GAs are fast, flexible, and easy to learn. Kheder et al. (2015) also applied the PPX crossover mechanism in an objective function from the viewpoint of maintenance. These researchers have explored the DSP problem with the GAs concept. In this study, the authors attempt to establish a new genetic algorithm to solve the DSP-related problems. In the past, the GAs mechanism proposed by Kongar and Gupta (2006) (denoted as K&G) was highly acclaimed. In this study, therefore, the verification of the newly proposed GA is mainly based on the comparison with their results.

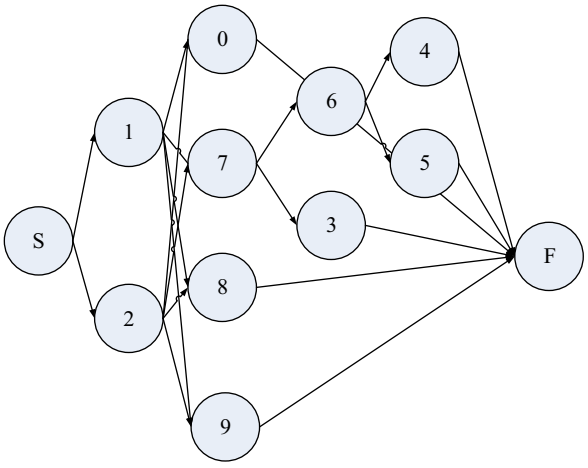
Another issue is that the random searching method in traditional GAs often faces the problem of a large range and is not effective in solutions. To improve the quality of the solution, this study proposes a new algorithm based on the block concept, defined as block-based GAs in this paper. Such a method helps to skip the local solution and processes the search task on the basis of an objective function and penalty function matrix. As a result, the fitness function is able to guide the search in a better direction (blocks of better scores), thus making it easier to search for the optimal solution and improve the efficiency of the algorithm. In addition, the present study is focused on the shortest path algorithms for the comparison of non-genetic algorithms. Among related methods, Dijkstra's algorithms are more efficient. Consequently, it is chosen as the method for verification of the method proposed in this study (Zhong et al., 2011).

In Section 2, the tool, direction, and priority relationship in disassembly are discussed. Also discussed in this section are K&G's algorithms and Dijkstra's algorithms. The methodology proposed in this study, block-based GAs, is delineated in Section 3. In Section 4, tests and comparisons of practical examples are offered to verify the suitability and effectiveness of our proposed approach. Section 5 provides our concluding remarks.

2. Construct the disassembly sequence planning

The definitions of the variables used in the algorithm are listed and described below:

Notes	Definition
N	number of parts in a product
J	index for component
seq	index for disassembly sequence
$P_{j,seq}$	penalty function (disassembly component j in sequence seq)
$D_{j,seq}$	score in the change of disassembly direction (disassembly component j in sequence seq)
$T_{j,seq}$	the score in the change of disassembly method or tool (disassembling component j in sequence seq)
MS	the fitness function value
M	the number of chromosomes
m	the amount of mutation
$Block\text{-}size$	the size of a block



(a)

Parts	0	1	2	3	4	5	6	7	8	9
Direction	$+Y$	$+X$	$+X$	$+Z$	$-Z$	$-Z$	$-Y$	$-X$	$-Z$	$-Y$
Tool	T_1	T_2	T_2	T_1	T_2	T_2	T_1	T_1	T_2	T_1

(b)

Fig. 1. The disassembly priority precedence relationships in Kongar and Gupta's example (a) Disassembly priority precedence graph (b) parts disassembly information.

2.1. Matrix for the disassembly directions and disassembly tools

In the disassembly direction, the present study adopts the same design as K&G, mainly six directions, ± X, ±Y, ±Z. For a change of 90° in the disassembly direction, a score of 1 is given; a score of 2 is assigned for a change of 180°. The scores for the disassembly directions are shown in Appendix.

For the disassembly method, the idea of the change of disassembly tools is used to match the practical conditions. A score of 1 will be assigned for a change of disassembly tool. For cases in which four disassembly tools will be used, the scores for changes of the disassembly tools are listed in Appendix.

In the example used in K&G's study (Fig. 1), there are 10 parts, and S and F are set as the virtual front and rear parts.

2.2. The design of the objective function

Before disassembly sequence planning, the engineering attributes of product parts should be clarified. The change of part attributes may cause waste of cost and affect the evaluation of disassembly sequence planning. This study adopts the same method as K&G with the penalty function for problem solving. Fewer changes will result in a lower penalty function value. The penalty function value is calculated according to the change in disassembly direction and disassembly tools or methods. Formula (1) shows that the penalty function value is the summation of the scores from the change in disassembly direction and disassembly tools:

$$P_{j,seq} = D_{j,seq} + T_{j,seq} \tag{1}$$

Given that the disassembly sequence planning is $P_{6,0} \rightarrow P_{4,1}$, the direction of $P_{6,0}$ is –Y; the direction of $P_{4,1}$ is –Z; and the direction will be changed from +Y to –Z. A score of 1 is assigned for a direction of 90° or a change of disassembly tools or methods. According to Formula (1), $P_{6,0}=D_{6,0}+T_{6,0}=1+1=2$. Table 1 lists the Penalty Function matrix for the combination of the priority precedence order of all parts. In this study, the objective function of disassembly sequence planning is set to be the summation of the penalty function values of all parts. A lower penalty function value indicates

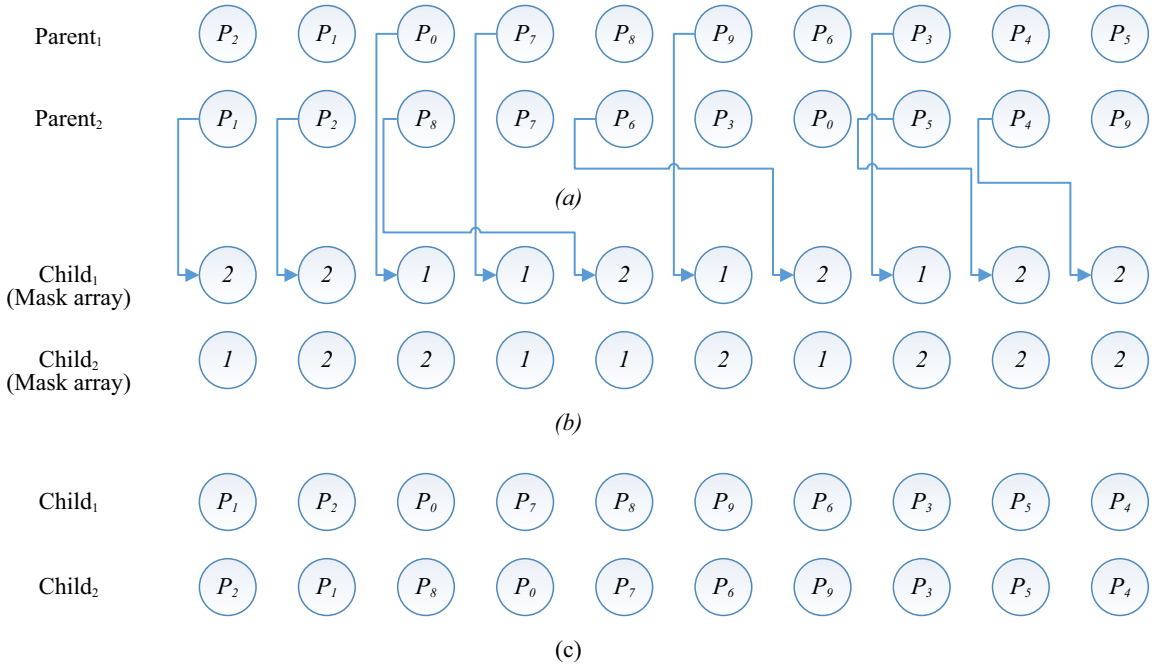


Fig. 2. The PPX crossover mechanism.

Table 1
Penalty function matrix in K&G's example.

No.	0	1	2	3	4	5	6	7	8	9
0	0	2	2	1	2	2	2	1	2	2
1	2	0	0	2	1	1	2	3	1	2
2	2	0	0	2	1	1	2	3	1	2
3	1	2	2	0	3	3	1	1	3	1
4	2	1	1	3	0	0	2	2	0	2
5	2	1	1	3	0	0	2	2	0	2
6	2	2	2	1	2	2	0	1	2	0
7	1	3	3	1	2	2	1	0	2	1
8	2	1	1	3	0	0	2	2	0	2
9	2	2	2	1	2	2	0	1	2	0

lower cost wastes in the disassembly process. It is illustrated in Formula (2).

$$MS = \sum_{seq=0}^{n-2} P_{j,seq}, \ \forall\ j, j = 0, \ \ldots, \ n-1 \tag{2}$$

In the example of Fig. 1, suppose the disassembly sequence planning is set as $P_2 \rightarrow P_1 \rightarrow P_0 \rightarrow P_8 \rightarrow P_7 \rightarrow P_6 \rightarrow P_3 \rightarrow P_5 \rightarrow P_9 \rightarrow P_4$. From the matrix of penalty function, the penalty function value of $P_{2,0}$, $P_{2,0} \rightarrow P_{1,1}$ is *2*; the penalty function value of $P_{1,1}$, $P_{1,1} \rightarrow P_{0,2}$, is *2*; the penalty function value of $P_{0,2}$, $P_{0,2} \rightarrow P_{8,3}$, is *2*; the penalty function value of $P_{8,3}$, $P_{8,3} \rightarrow P_{7,4}$, is *2*; the penalty function value of $P_{7,4}$, $P_{7,4} \rightarrow P_{6,5}$, is *0*; the penalty function value of $P_{6,5}$, $P_{6,5} \rightarrow P_{3,6}$, is *1*; the penalty function value of $P_{3,6}$, $P_{3,6} \rightarrow P_{5,7}$, is *3*; the penalty function value of $P_{5,7}$, $P_{5,7} \rightarrow P_{9,8}$, is *2*; and the penalty function value of $P_{9,8}$, $P_{9,8} \rightarrow P_{4,9}$, is *2*. From Formula (2), the objective function can be calculated as $MS = P_{2,0} + P_{1,1} + P_{0,2} + P_{8,3} + P_{7,4} + P_{6,5} + P_{3,6} + P_{5,7} + P_{9,8} = 2+2+2+2+0+1+3+2+2 = 16$. In other words, the fitness function value is *16*.

2.3. Kongar and Gupta's genetic algorithms

In order to maintain feasible solutions during the evolution process, K&G (2006) take into consideration the priority sequential order in the procedures of parent generation, crossover, and

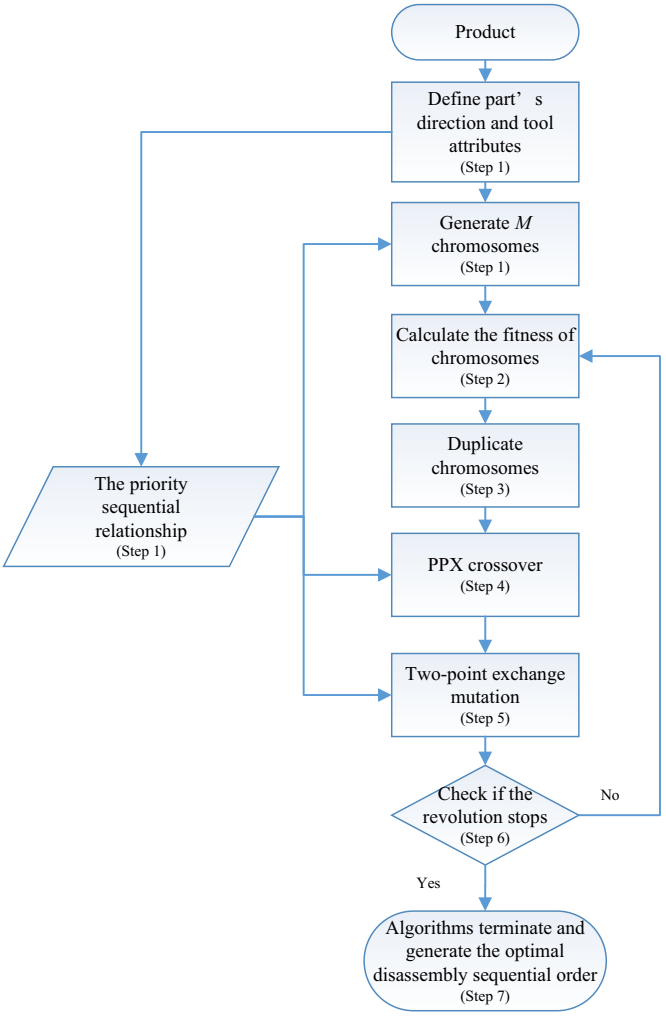


Fig. 3. The procedure in Kongar and Gupta's GAs.

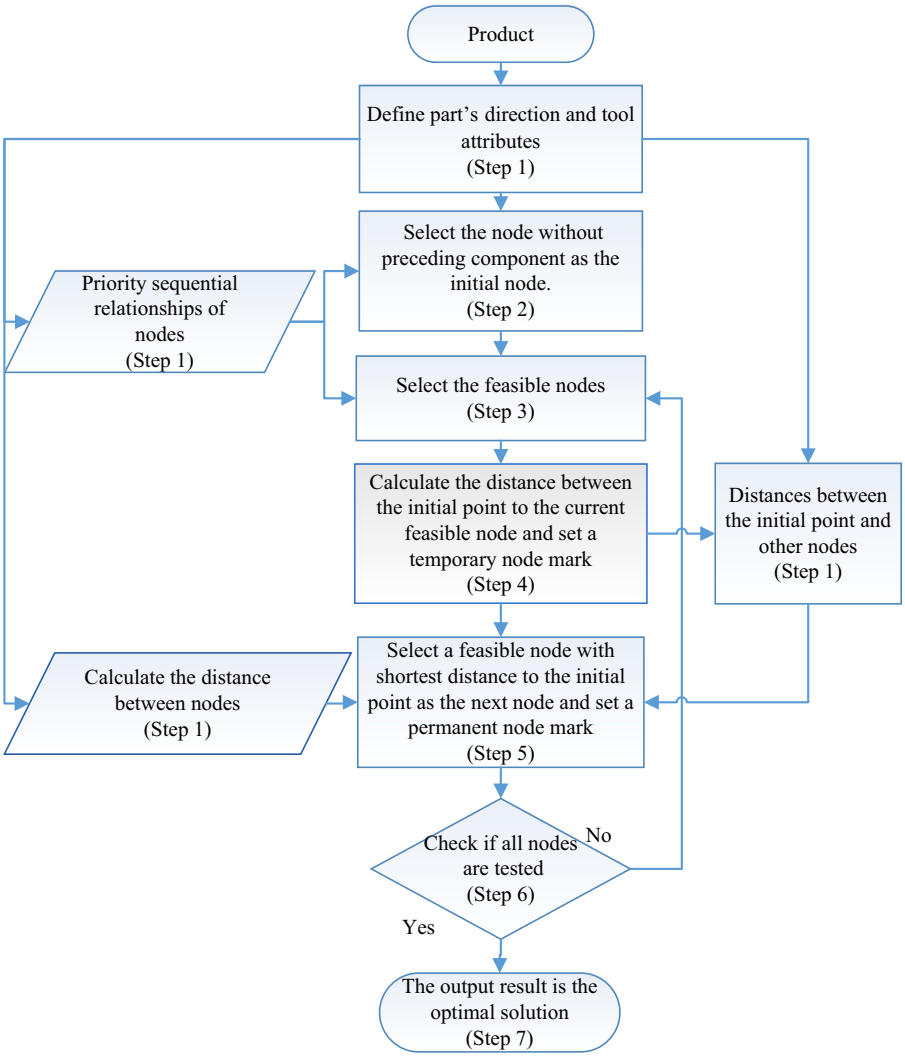


Fig. 4. Dijkstra's algorithms.

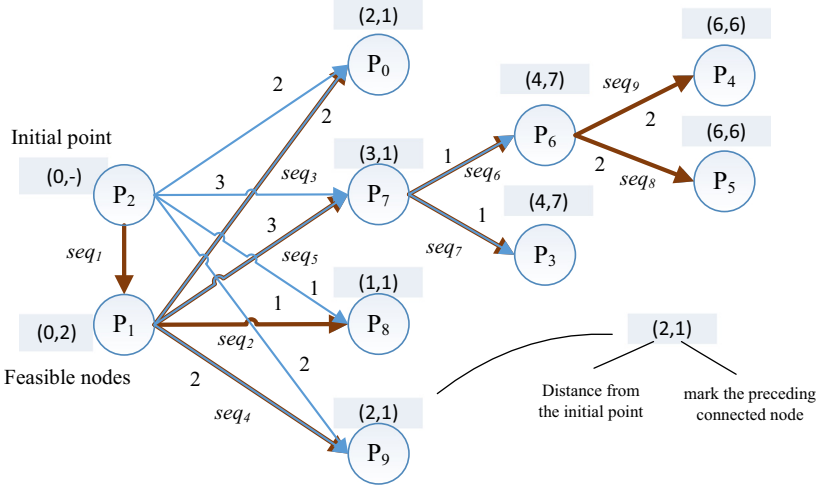


Fig. 5. The example in Dijkstra's algorithms.

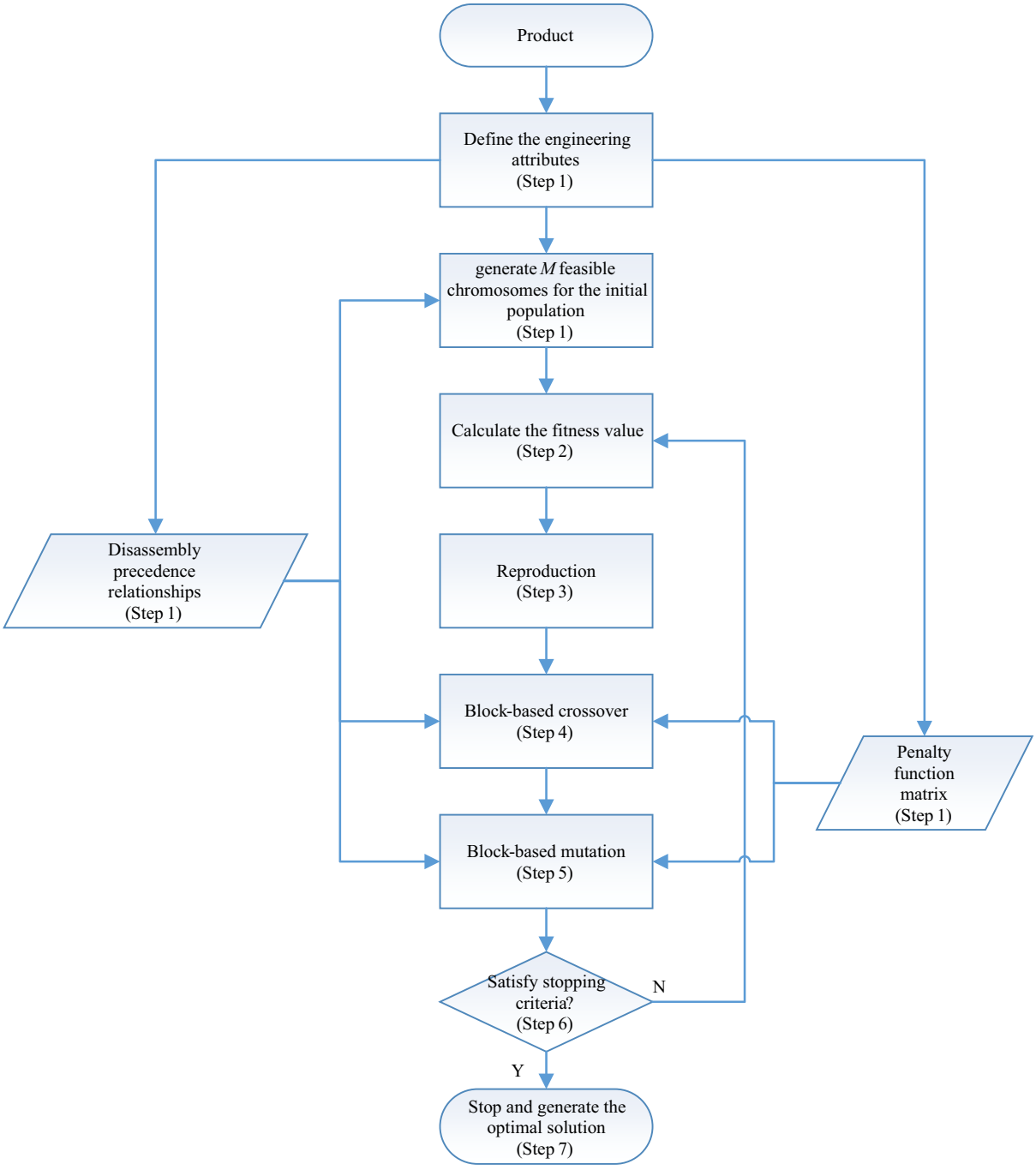


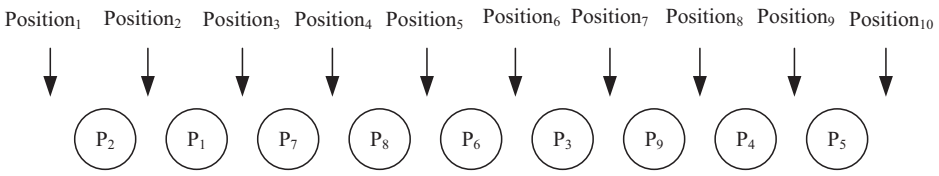
Fig. 6. Procedure of block-based genetic algorithms.

mutation. In their GAs, the tournament method is used to select fine chromosomes in a generation.

K&G's revolution mechanism is referred to as the PPX crossover method. Its distinguishing feature is the selection of two chromosomes with random coding for new chromosomes. Take Fig. 1 as an example. Two chromosomes, $Parent_1$ and $Parent_2$, are first selected, as shown in Fig. 2(a). Then two chromosomes, $Child_1$ and $Child_2$, are generated and randomly coded as No. 1 and No. 2 in the mask array. They represent the parent number from which the gene is selected, as shown in Fig. 2(b). Finally, genetic codes are inserted into these chromosomes according to the coding order. If the outcome is repeated or does not correspond to the priority sequential order, then the method jumps to next genetic code to maintain chromosome flexibility (Fig. 2(c)). In terms of the mutation method, K&G adopt the random two-point exchange method. The main procedure of K&G's GAs is listed in Fig. 3.

2.4. Dijkstra's algorithms

Dijkstra's algorithms are adjusted to fit the DSP conditions (Rardin, 1998). Dijkstra's Algorithms select the next nodes according to their shortest distance to the next code and add the priority sequential relationships in the procedure to maintain the feasibility of the solutions, as can be seen in Fig. 4. Dijkstra's Algorithms are illustrated in Fig. 1 and the major procedure is to choose the shortest distance between the initial point and current node.



(a)

Position	3	4	5	6	7	8	9	10
Score	3	3	4	3	3	4	4	4

(b)

Fig. 7. Positions of the chromosomes (a) Positions of the $Offspring_1$ (b) The scores of $Offspring_1$.

Table 2
Comparison of algorithms in four examples.

	Method	Number of Times	Sec/time	Average score	Optimal score	Average number of generations
Kongar and Gupta	Kongar and Gupta's GAs	10	0.124	7.2	7	21
	Block-based GAs	10	0.3	7	7	7
	PPX+block	10	0.21	7	7	22
	Shortest path	10	0.002	10	10	
Stapler	Kongar and Gupta's GAs	10	2.7	15.6	14	73
	Block-based GAs	10	4.1	12	12	41
	PPX+block	10	1.9	12.4	12	51
	Shortest path	10	0.0051	19.4	17	
printer	Kongar and Gupta's GAs	10	83	30.2	26	202
	Block-based GAs	10	54	14.4	14	103
	PPX+block	10	56	16.3	14	257
	Shortest path	10	0.0271	53.6	49	
virtual 150 nodes	Kongar and Gupta' GAs	10	1154	136	120	702
	Block-based GAs	10	612	55	53	1099
	PPX+block	10	1088	57	56	951
	Shortest path	10	0.1737	166	160	

Note: Average number of generations stands for the mean generation number in ten times of revolution.

Step 1: Define the part's direction, tool, and priority sequential order according to the product description, and calculate the penalty (distance) between the initial point and other nodes (see Fig. 5).
Step 2: Randomly select an initial node from those without preceding component. In Fig. 5, only P_1 and P_2 belong to this type. Currently, P_2 is selected.
Step 3: Search for the nodes that match the priority sequential order; in Fig. 1, only P_1 is suitable whereas the other nodes do not match the priority sequential order.
Step 4: Calculate the distance between the initial point to the current feasible node and set a temporary node mark.
Step 5: From all nodes that can be connected, select the node closest to the initial point. Currently, only P_1 can be connected. The distance between initial point (P_2) and P_1 is 0; therefore, P_1 is selected as the next node (permanent n ode), as the sequential order shown in Fig. 5.
Step 6: Check if all nodes are tested; then proceed to Step 3.
Step 3: Identify the nodes that match the current priority sequential order. In this case, four nodes, P_0, P_7, P_8, and P_9, can be connected but the others do not match the priority sequential order.
Step 4: Calculate the distance between the initial point and the current feasible node and set a temporary node mark.
Step 5: From all nodes that can be connected, select the node closest to the initial point. Since the distance between the initial point (P_2) and P_8 is 1, P_8 is selected for the next node, as the sequential order shown in Fig. 5.
Step 6: Check all nodes and proceed to Step 3.
Step 7: Identify the optimal solution. In this case, the final best solution is $P_2 \rightarrow P_1 \rightarrow P_8 \rightarrow P_0 \rightarrow P_9 \rightarrow P_7 \rightarrow P_6 \rightarrow P_3 \rightarrow P_5 \rightarrow P_4$.

3. The Block-based genetic algorithms

3.1. The procedure in Block-based genetic algorithms

The main difference between Block-based GAs and K&G GAs lies in the introduction of the penalty function matrix in the crossover and mutation mechanism. Fig. 6 illustrates the key procedure of Block-based GAs:

Step 1: The chromosome is coded by decimal digits to define the engineering attributes of the parts. Through the limit of the priority sequential order of parts, *M* feasible chromosomes are generated as the initial Parent.
Step 2: Calculate the Fitness function value of each chromosome for the evaluation of the chromosome.
Step 3: By the roulette method, reproduce an equal number of the Parent's chromosomes for the chromosomes in the next generation Parent.
Step 4: Process the block-based crossover mechanism.
Step 5: Process the block-based mutation mechanism.
Step 6: Check if it is time to terminate the evolution process. In this study, the maximum number of generations serves as the termination condition. If the termination condition occurs, proceed to Step 7; otherwise, proceed to Step 2.

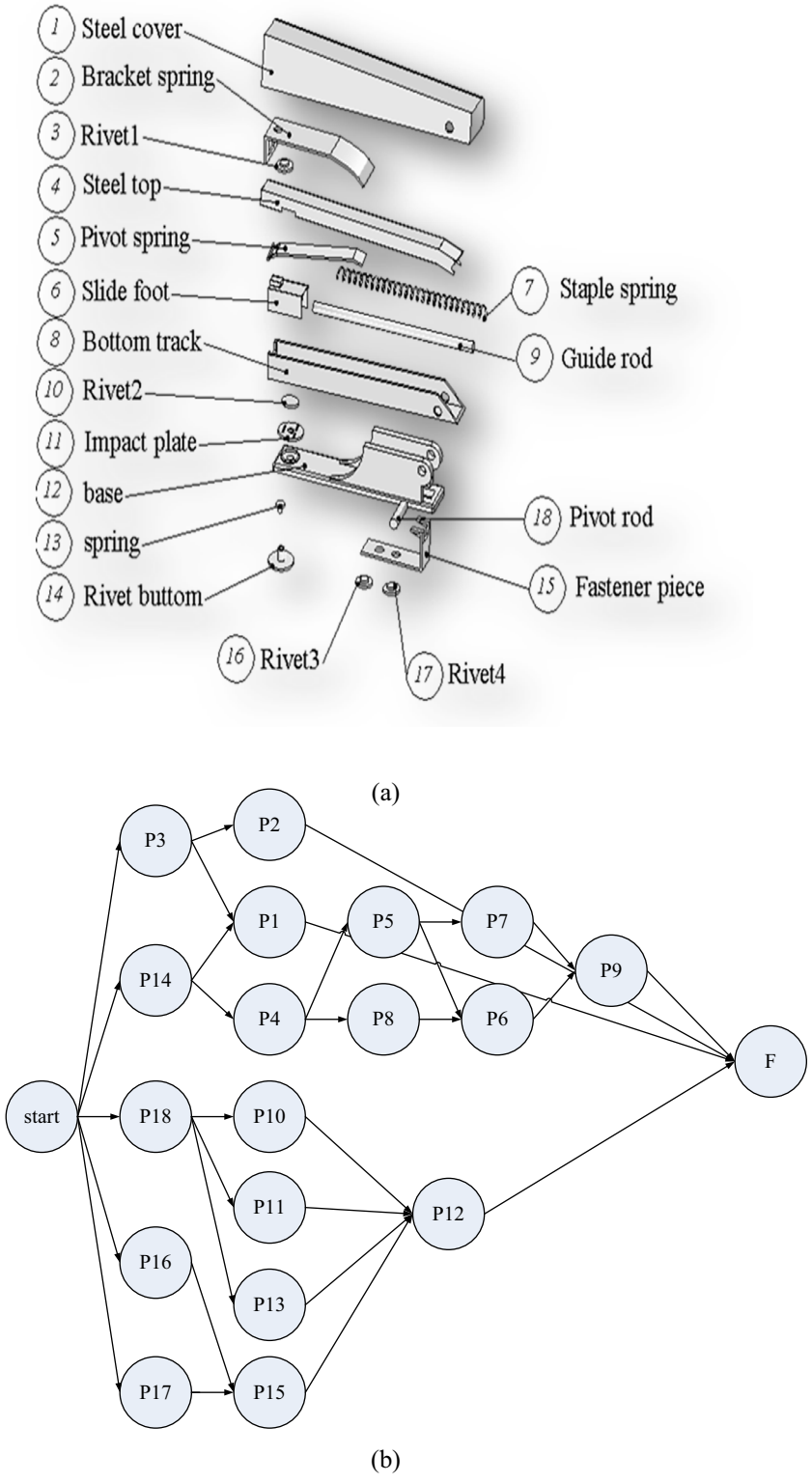
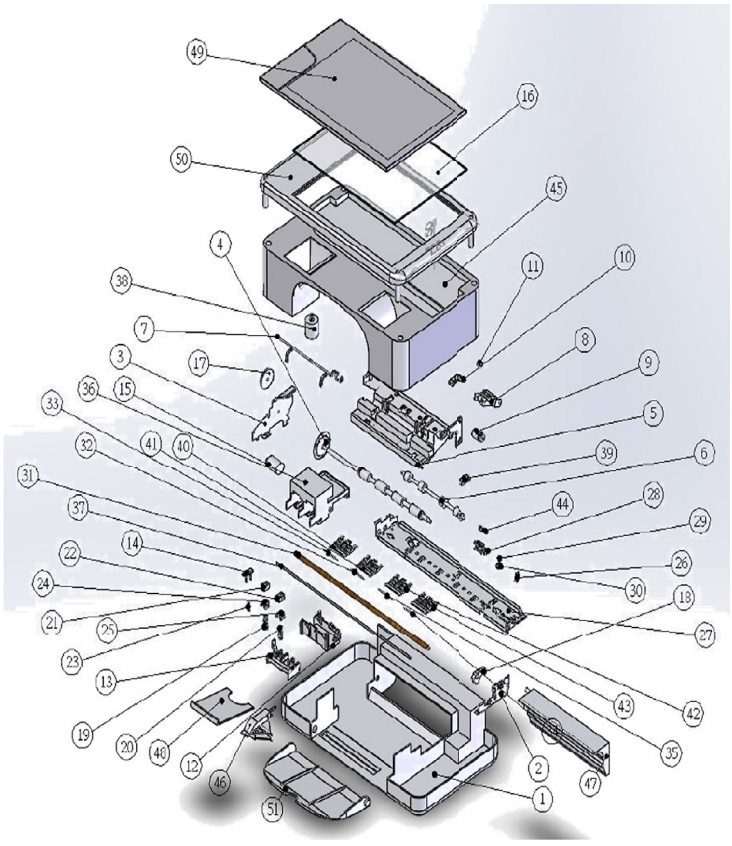


Fig. 8. Stapler example (a) product 3D diagram (b) disassembly priority precedence graph.

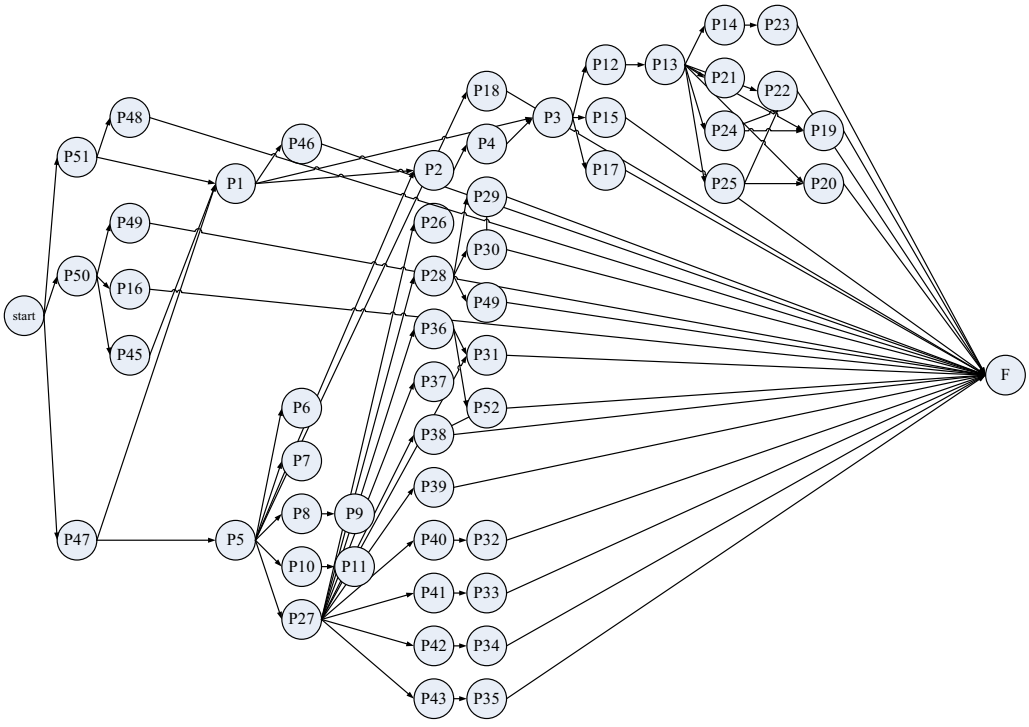
Step 7: Stop the algorithms and generate the optimal or near optimal disassembly sequential order.

3.2. Block-based crossover mechanism

There are two stages in the block-based crossover mechanism. In Stage 1, a block in the chromosome is reserved after the crossover operation, from which the block with best genes is maintained in the crossover procedure. In Stage 2, the genetic codes of the chromosome are exchanged for the next generation. The reservation of blocks and exchange of missing genetic codes ensures that the genes in the chromosome are good and offer a better opportunity to skip the local optimal solution.



(a)



(b)

Fig. 9. Printer example (a) the product 3D diagram for Printer (b) the priority precedence graph.

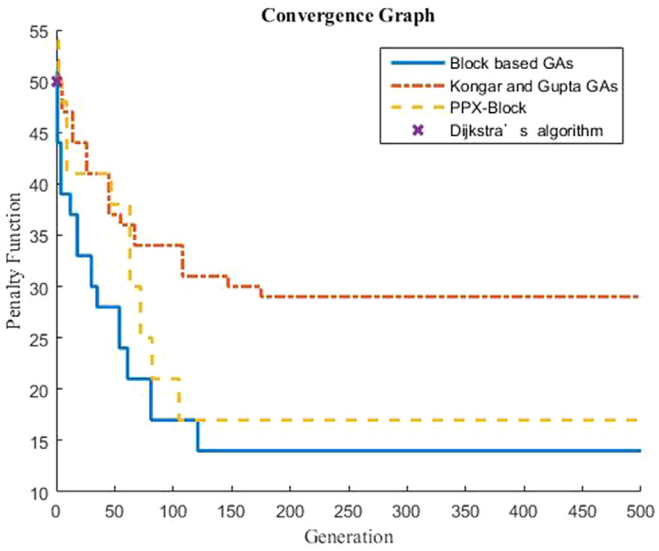


Fig. 10. The convergence graph of Printer.

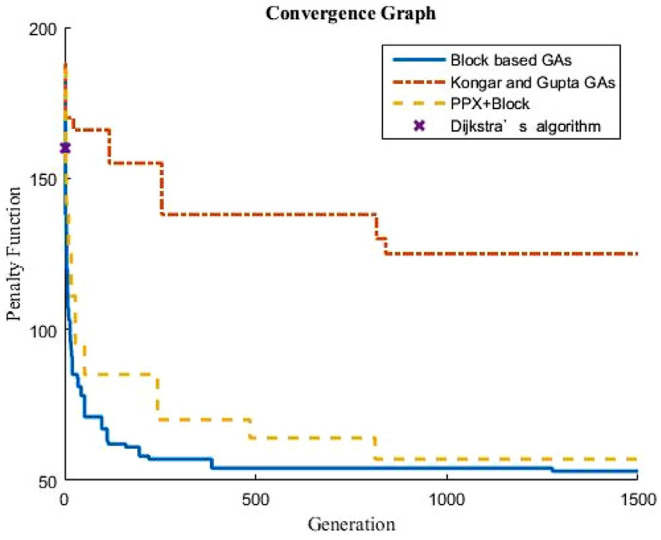


Fig. 12. The convergence graph for the virtual 150 nodes example.

3.2.1. The reservation of block

The purpose of Stage 1 is to reserve the block in the chromosome whose fitness function value is best. The size of a block can be *0%–100%* of the chromosome, which is generated randomly. If the size of the chromosome is *n*, and the *Block-size* is randomly generated, then there will be $(n\text{- }Block\text{-}size+1)$ blocks. From the fitness function values of all blocks, the block with the best the

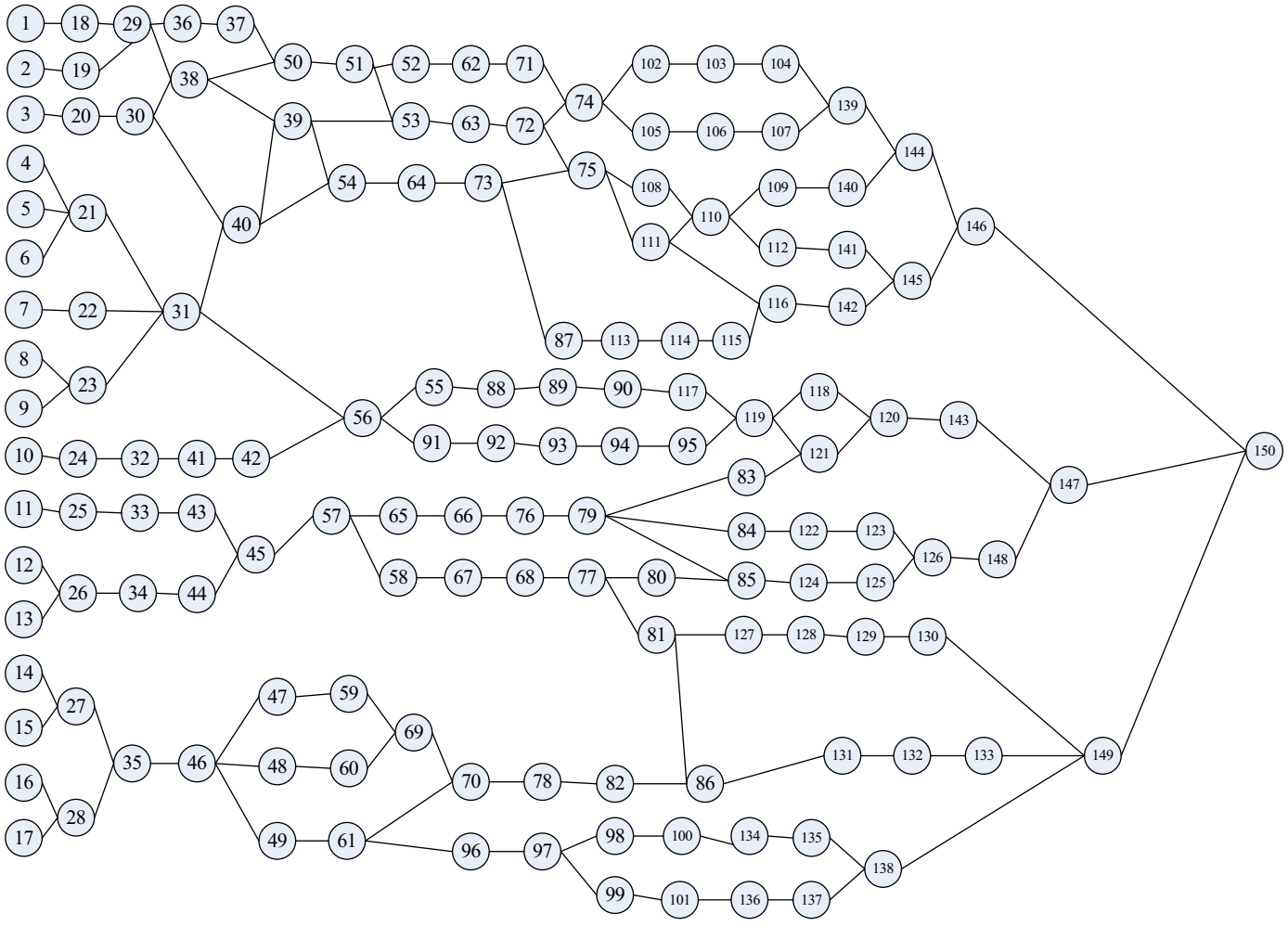


Fig. 11. The priority precedence graph for the virtual 150 nodes example (b) the convergence graph.

fitness function value will be reserved. The steps to keep such a block are described below:

Step 4.1: From the Parent, randomly select two chromosomes, $Parent_1$ and $Parent_2$, the *Block-size* of each of which is generated randomly.
Step 4.2: Divide each chromosome into (n - *Block-size* + 1) blocks.
Step 4.3: Calculate the fitness function values of all blocks.
Step 4.4: The block whose fitness function value is best is the optimal block. If more than one block has the best fitness function value, then randomly select one of these blocks.

3.2.2. Exchange the genetic codes in chromosome

The steps of Stage 2 help ensure that after the crossover operation, the offspring generation chromosomes also meet the limit conditions. Therefore, the limit conditions of the disassembly problem should be taken into consideration in reproducing and exchanging the genetic codes of the chromosome. The key steps for the exchange of the genetic codes in the chromosome are described as follows:

Step 4.5: Directly reproduce an offspring generation chromosome $Offspring_2$ from the parent generation chromosome $Parent_1$ and directly reproduce the offspring generation chromosome $Offspring_1$ from chromosome $Parent_2$. Reserve the blocks of the parent generation chromosomes $Parent_1$ and $Parent_2$, and directly reproduce the offspring generation chromosomes $Offspring_1$ and $Offspring_2$ in the relative positions of the chromosome.
Step 4.6: Search for the repeated and missing genetic codes in the offspring generation chromosomes $Offspring_1$ and $Offspring_2$.
Step 4.7: Delete the repeated genetic codes in the offspring generation chromosomes $Offspring_1$ and $Offspring_2$.
Step 4.8: Search for the possible positions to insert the missing genetic codes.
Step 4.9: Calculate the fitness function values of all possible positions for the insertion of missing genetic codes.
Step 4.10: Select the best position for the insert of missing genetic codes.
Step 4.11: Individually insert the missing genetic codes into the offspring generation chromosomes $Offspring_1$ and $Offspring_2$.

3.3. Block-based mutation mechanism

The block-based mutation mechanism is mainly designed to skip the local optimal solution. In one-point or two-point random exchange mutation methods, it is difficult to skip the local optimal solution in highly complicated conditions. In light of this, the block-based mutation mechanism adopts a multiple-point method to increase the effectiveness of skipping the local optimal solution. In highly complicated conditions, significant differences will exist before and after the mutation operation. The traditional mutation belongs to random algorithms, while the block-based mutation mechanism is goal-oriented. The key steps of the evolution process in the block-based algorithms are described below:

Step 5.1: Randomly select a point from the chromosome for the mutation point and temporarily remove it from the chromosome.
Step 5.2: Find the range for the insertion of the mutation point, confirming that the generated solutions are feasible ones.
Step 5.3: Calculate the additional fitness function values for all mutation points in the possible range.
Step 5.4: Find the best fitness function value for the insertion of the mutation point.
Step 5.5: Repeat Step 5.1–Step 5.4 m times (m is the mutation amount) and finish the mutation mechanism.

Fig. 1 illustrates the algorithm steps by using K&G's example.

Step 1: Define the disassembly direction, disassembly tool, and the disassembly sequential relationships. Calculate the penalty function matrix from the engineering information of the disassembly direction, and disassembly tools, which serves as the basis for the block-based mutation mechanism.
Step 2: Calculate the fitness function value for each chromosome for the chromosome evaluation.
Step 3: According to the fitness function value, reproduce the same number of the parent's chromosomes by the roulette method. In the example, M is set to *20*.
Step 4: Generate blocks and exchange genetic codes for the chromosomes decided for the crossover operation. Suppose we have a chromosome $Parent_1$ with a disassembly sequence planning of $P_1 \rightarrow P_2 \rightarrow P_0 \rightarrow P_7 \rightarrow P_8 \rightarrow P_6 \rightarrow P_3 \rightarrow P_9 \rightarrow P_4 \rightarrow P_5$, the following steps show how to generate the reserved block and exchange the genetic codes.
Step 4.1: Randomly generate a block with a *Block-size* of *0%–100%* its length. The length of the block $n = 10$. If we want to generate a *60%* block, then the *Block-size=10*60%=6*.
Step 4.2: Divide the chromosome into (n - *Block-size* + 1) blocks. In total, there will be $10 - 6 + 1 = 5$ blocks.
Step 4.3: The fitness function values of all blocks can be determined.
Step 4.4: The block whose fitness function value is best will be the block to reserve. The lowest fitness function value is *6*, so the block $P_8 \rightarrow P_6 \rightarrow P_3 \rightarrow P_9 \rightarrow P_4 \rightarrow P_5$ is the block to reserve in Stage 1.
Step 4.5: Suppose another chromosome $Parent_2$ for crossover operation is $P_2 \rightarrow P_1 \rightarrow P_8 \rightarrow P_7 \rightarrow P_0 \rightarrow P_9 \rightarrow P_3 \rightarrow P_0 \rightarrow P_4 \rightarrow P_5$. First reproduce the offspring generation chromosome $Offspring_1$ from $Parent_2$ and then reproduce the reserved block in $Parent_1$ at corresponding positions in $Offspring_1$. Now $Offspring_1$ turns into $P_2 \rightarrow P_1 \rightarrow P_8 \rightarrow P_7 \rightarrow P_8 \rightarrow P_6 \rightarrow P_3 \rightarrow P_9 \rightarrow P_4 \rightarrow P_5$.
Step 4.6: Check whether the offspring generation contains repeated genetic codes. We find that P_8 is repeated and P_0 is missing.
Step 4.7: Delete the repeated code P_8 so that $Offspring_1$ becomes $P_2 \rightarrow P_1 \rightarrow P_7 \rightarrow P_8 \rightarrow P_6 \rightarrow P_3 \rightarrow P_9 \rightarrow P_4 \rightarrow P_5$.
Step 4.8: Search for the range in $Offspring_1$ to insert P_0, and we find it is possible to insert P_0 from $Position_3$ to $Position_{10}$, as shown in Fig. 7(a).
Step 4.9: Calculate the score for possible insert positions. The calculation is based on the additional fitness function value in the position. For example, if it is inserted in $Position_3$, the additional fitness function value is $P_{1,0}+P_{0,7}=2+1=3$. For the insertion points of $Position_1$ and $Position_{10}$, the additional fitness function value is double. For objectivity in selecting insertion points, the additional position scores are listed in Fig. 7(b).
Step 4.10: From the score list, the best insertion points are $Position_3$, $Position_4$, $Position_6$, and $Position_7$, from which $Position_3$ is randomly selected.
Step 4.11: Insert the genetic code P_0; the $Offspring_1$ in the offspring generation will be: $P_2 \rightarrow P_1 \rightarrow P_0 \rightarrow P_7 \rightarrow P_8 \rightarrow P_6 \rightarrow P_3 \rightarrow P_9 \rightarrow P_4 \rightarrow P_5$. The exchange of genetic codes in Stage 2 is finished.
Step 5: Process the mutation mechanism for the chromosome in the following steps:

Step 5.1: According to the range of the mutation amount, randomly set the number of mutations to be generated to a certain amount. In this case, the random outcome is a mutation amount of 7.
Step 5.2: Randomly select a genetic code for the mutation point, and remove the mutation code from the chromosome.
Step 5.3: Insert the mutation point into the most suitable position by processing Step 4.8– Step 4.11.
Step 5.4: Repeat procedures Step 5.1–Step 5.3 7 times (7 is the mutation amount), and finish the mutation mechanism.
Step 6: Judge if it meets the termination condition. If yes, proceed to Step 7; otherwise, proceed to Step 2.
Step 7: Stop the algorithm and generate the optimal solution or the approaching optimal solution to the disassembly sequential order.

4. Sample verification

In this study, MATLAB R2015b is used for accomplishing the above-mentioned algorithms. The specification of the computer is Pentium(R) Dual-Core CPU E5200 @ 2.50 G and 2GB RAM. Four examples, including the example from Kongar and Gupta (2006), a Stapler, a Printer, and a Virtual example, are used for the comparisons of different algorithms. These examples are in increasing order in terms of the degree of complication. The engineering information for these examples is listed in the Appendix. In the tests, the number of experiments is 10 times; crossover rate *30%*, mutation rate *10%*. All of these related parameter values are decided after repeated tests.

In K&G's original design, the PPX crossover mechanism is able to solve the priority sequential problem. But its mutation method is the main cause of its low efficiency. In this study, the block-based mutation is able to solve the problem efficiently. As a result, it is suggested that PPX crossover and block-based mutation (PPX+Block) be combined for the verification. In this study, performances of four algorithms, including K&G, block-based GAs, PPX+Block, and Dijkstra's algorithms, are compared. In these algorithms, Dijkstra's algorithms do not have the effect of generation revolution, so its execution time is relatively short.

4.1. K&G example

The population size M is set to *20* and the maximum number of generations is *50*. The outcome of K&G's example is shown in Table 2. The result shows there is no big difference between the four algorithms, except that Dijkstra's algorithms do not produce a satisfactory result in such a simple example.

4.2. The stapler example

There are 18 parts in the stapler (Fig. 8(a)). The disassembly priority sequential relationship of the stapler parts is illustrated in Fig. 8(b). Moreover, the population size is set to 30, and the maximum number of generations is 200. The test result is listed in Table 2. It is found that the solutions of the block-based GAs and PPX+Block are better than those of the other two algorithms.

4.3. The printer example

There are 52 parts in the printer. The product 3D diagram and the disassembly priority sequential relationship of the printer parts are illustrated in Fig. 9(a) and (b). Furthermore, the population size is set to *100*, and the maximum number of generations is *500*. The convergence graphs are shown in Fig. 10. It is demonstrated that the solutions of block-based GAs and PPX+Block are relatively better; there is a moderate difference between K&G's and Dijkstra's algorithms (Table 2). In terms of the execution time, K&G's algorithms needs the longest time in the printer example.

4.4. A virtual complicated example with 150 nodes

To verify the influence of complicated examples on the algorithm, a virtual complicated example with 150 nodes is used (Fig. 11). In this example, the maximum number of generations is 1500 and the population size is 100. The convergence graphs are shown in Fig. 12. In terms of the solution quality, the block-based GAs are much better than those of the other algorithms (Table 2).

5. Conclusion

In this study, a block-based genetic algorithm is proposed to help solve the DSP problems. New mechanisms for crossover and mutation operations are developed to guide the evolution process for the objective direction. The block-based GA improves the solution search. Compared with Kongar and Gupta's GAs, the block-based genetic algorithm is more efficient in improving the solution quality as the complexity of the problems increases. However, the combination of the PPX mechanism and block-based mutation can effectively improve the problem. Dijkstra's algorithms have no generation revolution effect, the solution quality cannot improve progressively as GAs. But, because they are time-saving in execution. Dijkstra's algorithms can still be considered an appropriate method for occasions where the optimal solution is not required. In terms of the execution time, the block-based GAs will take a little more time in small cases because of the new mechanisms. But they work pretty well in complicated cases such as the printer and virtual examples.

On the basis of the current study, design for disassembly (DFD) is an issue worthy of exploration in the future. Furthermore, parallel operation will increase the efficiency of the GAs and significantly reduce the algorithm time for the application in products with more parts. This is another issue for future study.

Acknowledgements

This research was supported by the Ministry of Science and Technology of the Republic of China under grant number MOST 105-2410-H-167-007.

Appendix

Table A.1–Table A.6.

Table A.1
Engineering information for K&G example (a) change of disassembly direction (b) change of disassembly tools.

(a) Front/Rear	$+X$	$-X$	$+Y$	$-Y$	$+Z$	$-Z$
$+X$	0	2	1	1	1	1
$-X$	2	0	1	1	1	1
$+Y$	1	1	0	2	1	1
$-Y$	1	1	2	0	1	1
$+Z$	1	1	1	1	0	2
$-Z$	1	1	1	1	2	0

(b) Front/Rear	*T1*	*T2*	*T3*	*T4*
T1	0	1	1	1
T2	1	0	1	1
T3	1	1	0	1
T4	1	1	1	0

Table A.2
The disassembly directions and tools for stapler's parts.

Part no.	Disassembly Direction	Disassembly tool
1	$+Y$	T_3
2	$-Y$	T_1
3	$-Y$	T_2
4	$+Y$	T_3
5	$-Y$	T_4
6	$-X$	T_3
7	$+X$	T_3
8	$+Y$	T_3
9	$+X$	T_3
10	$-Y$	T_3
11	$+Y$	T_1
12	$-Y$	T_1
13	$-Y$	T_1
14	$-Y$	T_2
15	$-Y$	T_4
16	$-Y$	T_2
17	$-Y$	T_2
18	$-Z$	T_3

Table A.3
Penalty function matrix in stapler's example.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	0	3	3	0	3	1	1	0	1	2	1	3	3	3	3	3	3	1
2	3	0	1	3	1	2	2	3	2	1	2	0	0	1	1	1	1	2
3	3	1	0	3	1	2	2	3	2	1	3	1	1	0	1	0	0	2
4	0	3	3	0	3	1	1	0	1	2	1	3	3	3	3	3	3	1
5	3	1	1	3	0	2	2	3	2	1	3	1	1	1	0	1	1	2
6	1	2	2	1	2	0	2	1	2	1	2	2	2	2	2	2	2	1
7	1	2	2	1	2	2	0	1	0	1	2	2	2	2	2	2	2	1
8	0	3	3	0	3	1	1	0	1	2	1	3	3	3	3	3	3	1
9	1	2	2	1	2	2	0	1	0	1	2	2	2	2	2	2	2	1
10	2	1	1	2	1	1	1	2	1	0	3	1	1	1	1	1	1	1
11	1	2	3	1	3	2	2	1	2	3	0	2	2	3	3	3	3	2
12	3	0	1	3	1	2	2	3	2	1	2	0	0	1	1	1	1	2
13	3	0	1	3	1	2	2	3	2	1	2	0	0	1	1	1	1	2
14	3	1	0	3	1	2	2	3	2	1	3	1	1	0	1	0	0	2
15	3	1	1	3	0	2	2	3	2	1	3	1	1	1	0	1	1	2
16	3	1	0	3	1	2	2	3	2	1	3	1	1	0	1	0	0	2
17	3	1	0	3	1	2	2	3	2	1	3	1	1	0	1	0	0	2
18	1	2	2	1	2	1	1	1	1	1	2	2	2	2	2	2	2	0

Table A.4
The disassembly directions and tools for printer's parts.

Part no.	Disassembly Direction	Disassembly tool	Part no.	Disassembly Direction	Disassembly tool
1	$+Y$	T_2	27	$+Y$	T_1
2	$+Y$	T_2	28	$+Y$	T_2
3	$+Y$	T_1	29	$-Y$	T_1
4	$-Y$	T_1	30	$+Y$	T_1
5	$+X$	T_1	31	$-Y$	T_2
6	$+X$	T_1	32	$+X$	T_1
7	$-X$	T_1	33	$+X$	T_1
8	$+Y$	T_1	34	$+X$	T_1
9	$+X$	T_1	35	$+X$	T_1
10	$+X$	T_1	36	$+X$	T_1
11	$+X$	T_1	37	$-Y$	T_1
12	$-X$	T_1	38	$-Y$	T_2
13	$+Y$	T_1	39	$+X$	T_1
14	$+Y$	T_1	40	$+Y$	T_1
15	$+X$	T_1	41	$+Y$	T_1
16	$-Y$	T_2	42	$+Y$	T_1
17	$-Y$	T_1	43	$+Y$	T_1
18	$-Y$	T_1	44	$-Y$	T_1
19	$-Y$	T_1	45	$+Y$	T_2
20	$+Y$	T_1	46	$+Z$	T_1
21	$+Y$	T_2	47	$-Z$	T_1
22	$+Y$	T_2	48	$+Z$	T_1
23	$+Y$	T_1	49	$+Y$	T_1
24	$+Y$	T_2	50	$+Y$	T_2
25	$+Y$	T_2	51	$-Z$	T_1
26	$-Y$	T_1	52	$+Y$	T_1

Table A.5
Penalty function matrix in printer's example.

	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	41	42	43	44	45	46	47	48	49	50	51	52
1	0	0	1	3	2	2	2	1	2	2	2	2	1	1	2	2	3	3	3	1	0	0	1	0	0	3	1	0	3	1	2	2	2	2	2	2	3	2	2	1	1	1	1	3	0	2	2	2	1	0	2	1
2	0	0	1	3	2	2	2	1	2	2	2	2	1	1	2	2	3	3	3	1	0	0	1	0	0	3	1	0	3	1	2	2	2	2	2	2	3	2	2	1	1	1	1	3	0	2	2	2	1	0	2	1
3	1	1	0	2	1	1	1	0	1	1	1	1	0	0	1	3	2	2	2	0	1	1	0	1	1	2	0	1	2	0	3	1	1	1	1	1	2	3	1	0	0	0	0	2	1	1	1	1	0	1	1	0
4	3	3	2	0	1	1	1	2	1	1	1	1	2	2	1	1	0	0	0	2	3	3	2	3	3	0	2	3	0	2	1	1	1	1	1	1	0	1	1	2	2	2	2	0	3	1	1	1	2	3	1	2
5	2	2	1	1	0	0	2	1	0	0	0	2	1	1	0	2	1	1	1	1	2	2	1	2	2	1	1	2	1	1	2	0	0	0	0	0	1	2	0	1	1	1	1	1	2	1	1	1	1	2	1	1
6	2	2	1	1	0	0	2	1	0	0	0	2	1	1	0	2	1	1	1	1	2	2	1	2	2	1	1	2	1	1	2	0	0	0	0	0	1	2	0	1	1	1	1	1	2	1	1	1	1	2	1	1
7	2	2	1	1	2	2	0	1	2	2	2	0	1	1	2	2	1	1	1	1	2	2	1	2	2	1	1	2	1	1	2	2	2	2	2	2	1	2	2	1	1	1	1	1	2	1	1	1	1	2	1	1
8	1	1	0	2	1	1	1	0	1	1	1	1	0	0	1	3	2	2	2	0	1	1	0	1	1	2	0	1	2	0	3	1	1	1	1	1	2	3	1	0	0	0	0	2	1	1	1	1	0	1	1	0
9	2	2	1	1	0	0	2	1	0	0	0	2	1	1	0	2	1	1	1	1	2	2	1	2	2	1	1	2	1	1	2	0	0	0	0	0	1	2	0	1	1	1	1	1	2	1	1	1	1	2	1	1
10	2	2	1	1	0	0	2	1	0	0	0	2	1	1	0	2	1	1	1	1	2	2	1	2	2	1	1	2	1	1	2	0	0	0	0	0	1	2	0	1	1	1	1	1	2	1	1	1	1	2	1	1
11	2	2	1	1	0	0	2	1	0	0	0	2	1	1	0	2	1	1	1	1	2	2	1	2	2	1	1	2	1	1	2	0	0	0	0	0	1	2	0	1	1	1	1	1	2	1	1	1	1	2	1	1
12	2	2	1	1	2	2	0	1	2	2	2	0	1	1	2	2	1	1	1	1	2	2	1	2	2	1	1	2	1	1	2	2	2	2	2	2	1	2	2	1	1	1	1	1	2	1	1	1	1	2	1	1
13	1	1	0	2	1	1	1	0	1	1	1	1	0	0	1	3	2	2	2	0	1	1	0	1	1	2	0	1	2	0	3	1	1	1	1	1	2	3	1	0	0	0	0	2	1	1	1	1	0	1	1	0
14	1	1	0	2	1	1	1	0	1	1	1	1	0	0	1	3	2	2	2	0	1	1	0	1	1	2	0	1	2	0	3	1	1	1	1	1	2	3	1	0	0	0	0	2	1	1	1	1	0	1	1	0
15	2	2	1	1	0	0	2	1	0	0	0	2	1	1	0	2	1	1	1	1	2	2	1	2	2	1	1	2	1	1	2	0	0	0	0	0	1	2	0	1	1	1	1	1	2	1	1	1	1	2	1	1
16	2	2	3	1	2	2	2	3	2	2	2	2	3	3	2	0	1	1	1	3	2	2	3	2	2	1	3	2	1	3	0	2	2	2	2	2	1	0	2	3	3	3	3	1	2	2	2	2	3	2	2	3
17	3	3	2	0	1	1	1	2	1	1	1	1	2	2	1	1	0	0	0	2	3	3	2	3	3	0	2	3	0	2	1	1	1	1	1	1	0	1	1	2	2	2	2	0	3	1	1	1	2	3	1	2
18	3	3	2	0	1	1	1	2	1	1	1	1	2	2	1	1	0	0	0	2	3	3	2	3	3	0	2	3	0	2	1	1	1	1	1	1	0	1	1	2	2	2	2	0	3	1	1	1	2	3	1	2
19	3	3	2	0	1	1	1	2	1	1	1	1	2	2	1	1	0	0	0	2	3	3	2	3	3	0	2	3	0	2	1	1	1	1	1	1	0	1	1	2	2	2	2	0	3	1	1	1	2	3	1	2
20	1	1	0	2	1	1	1	0	1	1	1	1	0	0	1	3	2	2	2	0	1	1	0	1	1	2	0	1	2	0	3	1	1	1	1	1	2	3	1	0	0	0	0	2	1	1	1	1	0	1	1	0
21	0	0	1	3	2	2	2	1	2	2	2	2	1	1	2	2	3	3	3	1	0	0	1	0	0	3	1	0	3	1	2	2	2	2	2	2	3	2	2	1	1	1	1	3	0	2	2	2	1	0	2	1
22	0	0	1	3	2	2	2	1	2	2	2	2	1	1	2	2	3	3	3	1	0	0	1	0	0	3	1	0	3	1	2	2	2	2	2	2	3	2	2	1	1	1	1	3	0	2	2	2	1	0	2	1
23	1	1	0	2	1	1	1	0	1	1	1	1	0	0	1	3	2	2	2	0	1	1	0	1	1	2	0	1	2	0	3	1	1	1	1	1	2	3	1	0	0	0	0	2	1	1	1	1	0	1	1	0
24	0	0	1	3	2	2	2	1	2	2	2	2	1	1	2	2	3	3	3	1	0	0	1	0	0	3	1	0	3	1	2	2	2	2	2	2	3	2	2	1	1	1	1	3	0	2	2	2	1	0	2	1
25	0	0	1	3	2	2	2	1	2	2	2	2	1	1	2	2	3	3	3	1	0	0	1	0	0	3	1	0	3	1	2	2	2	2	2	2	3	2	2	1	1	1	1	3	0	2	2	2	1	0	2	1
26	3	3	2	0	1	1	1	2	1	1	1	1	2	2	1	1	0	0	0	2	3	3	2	3	3	0	2	3	0	2	1	1	1	1	1	1	0	1	1	2	2	2	2	0	3	1	1	1	2	3	1	2
27	1	1	0	2	1	1	1	0	1	1	1	1	0	0	1	3	2	2	2	0	1	1	0	1	1	2	0	1	2	0	3	1	1	1	1	1	2	3	1	0	0	0	0	2	1	1	1	1	0	1	1	0
28	0	0	1	3	2	2	2	1	2	2	2	2	1	1	2	2	3	3	3	1	0	0	1	0	0	3	1	0	3	1	2	2	2	2	2	2	3	2	2	1	1	1	1	3	0	2	2	2	1	0	2	1
29	3	3	2	0	1	1	1	2	1	1	1	1	2	2	1	1	0	0	0	2	3	3	2	3	3	0	2	3	0	2	1	1	1	1	1	1	0	1	1	2	2	2	2	0	3	1	1	1	2	3	1	2
30	1	1	0	2	1	1	1	0	1	1	1	1	0	0	1	3	2	2	2	0	1	1	0	1	1	2	0	1	2	0	3	1	1	1	1	1	2	3	1	0	0	0	0	2	1	1	1	1	0	1	1	0
31	2	2	3	1	2	2	2	3	2	2	2	2	3	3	2	0	1	1	1	3	2	2	3	2	2	1	3	2	1	3	0	2	2	2	2	2	1	0	2	3	3	3	3	1	2	2	2	2	3	2	2	3
32	2	2	1	1	0	0	2	1	0	0	0	2	1	1	0	2	1	1	1	1	2	2	1	2	2	1	1	2	1	1	2	0	0	0	0	0	1	2	0	1	1	1	1	1	2	1	1	1	1	2	1	1
33	2	2	1	1	0	0	2	1	0	0	0	2	1	1	0	2	1	1	1	1	2	2	1	2	2	1	1	2	1	1	2	0	0	0	0	0	1	2	0	1	1	1	1	1	2	1	1	1	1	2	1	1
34	2	2	1	1	0	0	2	1	0	0	0	2	1	1	0	2	1	1	1	1	2	2	1	2	2	1	1	2	1	1	2	0	0	0	0	0	1	2	0	1	1	1	1	1	2	1	1	1	1	2	1	1
35	2	2	1	1	0	0	2	1	0	0	0	2	1	1	0	2	1	1	1	1	2	2	1	2	2	1	1	2	1	1	2	0	0	0	0	0	1	2	0	1	1	1	1	1	2	1	1	1	1	2	1	1
36	2	2	1	1	0	0	2	1	0	0	0	2	1	1	0	2	1	1	1	1	2	2	1	2	2	1	1	2	1	1	2	0	0	0	0	0	1	2	0	1	1	1	1	1	2	1	1	1	1	2	1	1
37	3	3	2	0	1	1	1	2	1	1	1	1	2	2	1	1	0	0	0	2	3	3	2	3	3	0	2	3	0	2	1	1	1	1	1	1	0	1	1	2	2	2	2	0	3	1	1	1	2	3	1	2
38	2	2	3	1	2	2	2	3	2	2	2	2	3	3	2	0	1	1	1	3	2	2	3	2	2	1	3	2	1	3	0	2	2	2	2	2	1	0	2	3	3	3	3	1	2	2	2	2	3	2	2	3
39	2	2	1	1	0	0	2	1	0	0	0	2	1	1	0	2	1	1	1	1	2	2	1	2	2	1	1	2	1	1	2	0	0	0	0	0	1	2	0	1	1	1	1	1	2	1	1	1	1	2	1	1
40	1	1	0	2	1	1	1	0	1	1	1	1	0	0	1	3	2	2	2	0	1	1	0	1	1	2	0	1	2	0	3	1	1	1	1	1	2	3	1	0	0	0	0	2	1	1	1	1	0	1	1	0
41	1	1	0	2	1	1	1	0	1	1	1	1	0	0	1	3	2	2	2	0	1	1	0	1	1	2	0	1	2	0	3	1	1	1	1	1	2	3	1	0	0	0	0	2	1	1	1	1	0	1	1	0
42	1	1	0	2	1	1	1	0	1	1	1	1	0	0	1	3	2	2	2	0	1	1	0	1	1	2	0	1	2	0	3	1	1	1	1	1	2	3	1	0	0	0	0	2	1	1	1	1	0	1	1	0
43	1	1	0	2	1	1	1	0	1	1	1	1	0	0	1	3	2	2	2	0	1	1	0	1	1	2	0	1	2	0	3	1	1	1	1	1	2	3	1	0	0	0	0	2	1	1	1	1	0	1	1	0
44	3	3	2	0	1	1	1	2	1	1	1	1	2	2	1	1	0	0	0	2	3	3	2	3	3	0	2	3	0	2	1	1	1	1	1	1	0	1	1	2	2	2	2	0	3	1	1	1	2	3	1	2
45	0	0	1	3	2	2	2	1	2	2	2	2	1	1	2	2	3	3	3	1	0	0	1	0	0	3	1	0	3	1	2	2	2	2	2	2	3	2	2	1	1	1	1	3	0	2	2	2	1	0	2	1
46	2	2	1	1	1	1	1	1	1	1	1	1	1	1	1	2	1	1	1	1	2	2	1	1	2	2	1	1	2	1	1	2	1	1	1	1	1	2	1	1	1	1	1	1	2	0	2	0	1	2	2	1
47	2	2	1	1	1	1	1	1	1	1	1	1	1	1	1	2	1	1	1	1	2	2	1	2	2	1	1	2	1	1	2	1	1	1	1	1	1	2	1	1	1	1	1	1	2	2	0	2	1	2	0	1
48	2	2	1	1	1	1	1	1	1	1	1	1	1	1	1	2	1	1	1	1	2	2	1	2	2	1	1	2	1	1	2	1	1	1	1	1	1	2	1	1	1	1	1	1	2	0	2	0	1	2	2	1
49	1	1	0	2	1	1	1	0	1	1	1	1	0	0	1	3	2	2	2	0	1	1	0	1	1	2	0	1	2	0	3	1	1	1	1	1	2	3	1	0	0	0	0	2	1	1	1	1	0	1	1	0
50	0	0	1	3	2	2	2	1	2	2	2	2	1	1	2	2	3	3	3	1	0	0	1	0	0	3	1	0	3	1	2	2	2	2	2	2	3	2	2	1	1	1	1	3	0	2	2	2	1	0	2	1
51	2	2	1	1	1	1	1	1	1	1	1	1	1	1	1	2	1	1	1	1	2	2	1	2	2	1	1	2	1	1	2	1	1	1	1	1	1	2	1	1	1	1	1	1	2	2	0	2	1	2	0	1
52	1	1	0	2	1	1	1	0	1	1	1	1	0	0	1	3	2	2	2	0	1	1	0	1	1	2	0	1	2	0	3	1	1	1	1	1	2	3	1	0	0	0	0	2	1	1	1	1	0	1	1	0

Table A.6
The disassembly direction and tools for virtual 150 nodes example.

No.	Direction	Tool	No.	Direction	Tool	No.	Direction	Tool	No.	Direction	Tool	No.	Direction	Tool
1	$+X$	T_2	33	$+Y$	T_1	65	$+X$	T_1	97	$-Y$	T_2	129	$+Y$	T_1
2	$+X$	T_1	34	$-Y$	T_1	66	$+X$	T_1	98	$-Y$	T_2	130	$+Y$	T_1
3	$+X$	T_1	35	$-Y$	T_1	67	$+X$	T_1	99	$-Y$	T_2	131	$+Y$	T_1
4	$-X$	T_1	36	$+Y$	T_1	68	$-X$	T_1	100	$-Y$	T_2	132	$+Y$	T_1
5	$-X$	T_1	37	$+Y$	T_1	69	$+Y$	T_1	101	$+X$	T_2	133	$+Y$	T_1
6	$+X$	T_1	38	$+Y$	T_2	70	$+Y$	T_1	102	$+X$	T_2	134	$+Y$	T_1
7	$+X$	T_1	39	$-Y$	T_2	71	$+X$	T_1	103	$+Y$	T_2	135	$+Z$	T_1
8	$+X$	T_3	40	$-Y$	T_1	72	$+Y$	T_1	104	$+Y$	T_2	136	$+Z$	T_1
9	$+X$	T_1	41	$-Y$	T_1	73	$-Y$	T_2	105	$+Y$	T_1	137	$+Z$	T_1
10	$-X$	T_1	42	$+X$	T_1	74	$+Y$	T_2	106	$+Y$	T_1	138	$+Y$	T_1
11	$-Y$	T_1	43	$+X$	T_3	75	$+Y$	T_2	107	$+X$	T_1	139	$+Y$	T_3
12	$-Y$	T_1	44	$+X$	T_3	76	$-Y$	T_2	108	$+X$	T_1	140	$+Y$	T_3
13	$-Y$	T_2	45	$+X$	T_3	77	$+Y$	T_2	109	$+X$	T_1	141	$+Y$	T_2
14	$+Y$	T_2	46	$+X$	T_3	78	$-Y$	T_3	110	$+X$	T_1	142	$+X$	T_2
15	$+Y$	T_1	47	$+X$	T_1	79	$+X$	T_3	111	$+X$	T_1	143	$+X$	T_1
16	$+X$	T_1	48	$+X$	T_1	80	$+X$	T_3	112	$+X$	T_1	144	$+Y$	T_1
17	$+X$	T_1	49	$+X$	T_1	81	$+X$	T_1	113	$+X$	T_1	145	$+Y$	T_1
18	$+X$	T_1	50	$-Z$	T_1	82	$+X$	T_1	114	$+X$	T_3	146	$+Y$	T_1
19	$+X$	T_1	51	$+Y$	T_1	83	$-X$	T_1	115	$-Y$	T_3	147	$-Y$	T_1
20	$+Y$	T_1	52	$+Y$	T_2	84	$-X$	T_1	116	$-Y$	T_3	148	$-Y$	T_1
21	$+Y$	T_1	53	$+Y$	T_2	85	$+X$	T_1	117	$-Y$	T_3	149	$-Y$	T_1
22	$+Y$	T_1	54	$+Y$	T_3	86	$+X$	T_1	118	$-Y$	T_2	150	$-Y$	T_1
23	$+Y$	T_1	55	$+X$	T_3	87	$+X$	T_1	119	$-Y$	T_2			
24	$+Z$	T_1	56	$+X$	T_3	88	$-X$	T_2	120	$-Y$	T_2			
25	$+Z$	T_1	57	$-Z$	T_3	89	$-X$	T_2	121	$-Y$	T_2			
26	$+Z$	T_1	58	$-Z$	T_1	90	$-X$	T_2	122	$+Y$	T_2			
27	$+Y$	T_2	59	$-Z$	T_1	91	$-X$	T_2	123	$+Y$	T_2			
28	$+Y$	T_2	60	$+X$	T_2	92	$+Y$	T_1	124	$+Y$	T_2			
29	$+Y$	T_2	61	$+X$	T_2	93	$+Y$	T_1	125	$+Y$	T_2			
30	$+Y$	T_2	62	$+X$	T_2	94	$+Z$	T_1	126	$+Y$	T_2			
31	$+Y$	T_2	63	$+X$	T_2	95	$+Z$	T_1	127	$+Y$	T_2			
32	$+Y$	T_2	64	$-X$	T_2	96	$+Z$	T_2	128	$+Y$	T_1			

References

Adenso-Díaz, B., García-Carbajal, S., & Lozano, S. (2007). An efficient GRASP algorithm for disassembly sequence planning. *OR Spectrum, 29*(3), 535–549.

Ali Ilgin, M., & Gupta, S. M. (2010). Environmentally conscious manufacturing and product recovery (ECMPRO): A review of the state of the art. *Journal of Environmental Management, 91*, 563–591.

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*(1), 69–79.

Andrés, C., Lozano, S., & Adenso-Díaz, B. (2007). Disassembly sequence in a disassembly cell context. *Robotics and Computer Integrated Manufacturing, 23*, 690–695.

Ghandi, S., & Msaehian, E. (2015). Review and taxonomies of assembly and disassembly path planning problems and approaches. *Computer-Aided Design, 67-68*, 58–86.

Giudice, F., & Fargione, G. (2007). Disassembly planning of mechanical systems for service and recovery: A genetic algorithms based approach. *Journal of Intelligent Manufacturing, 18*, 313–329.

Go, T. F., Wahab, D. A., Rahman, M. N. Ab., Ramli, R., & Hussain, A. (2012). Genetically optimized disassembly sequence for automotive component reuse. *Expert System with Applications, 39*, 5409–5417.

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

Güngör, A., & Gupta, S. M. (2001). Disassembly sequence plan generation using a branch-and-bound algorithm. *International Journal of Production Research, 39*(3), 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*(11), 2435–2446.

Homen De Mello, L. S., & Sanderson, A. C. (1991). A correct and complete algorithm for the generation of mechanical assembly sequences. *IEEE Transaction on Robotics and Automation, 7*, 228–240.

Huang, H. H., Wang, M. H., & Johnson, M. R. (2000). Disassembly sequence generation using a neural network approach. *Journal of Manufacturing Systems, 19*(2), 73–82.

Hui, W., Dong, X., & Guanghong, D. (2008). A genetic algorithm for product disassembly sequence planning. *Neurocomputing, 71*(13), 2720–2726.

Kheder, M., Trigui, M., & Aifaoui, N. (2015). Disassembly sequence planning based on a genetic algorithm. *Proc IMechE Part C Journal of Mechanical of Engineering Science, 3*, 1–10.

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

Lambert, A. J. D. (1999). Linear programming in disassembly/clustering sequencing generation. *Computer and Industrial Engineering, 36*, 723–738.

Lambert, A. J. D. (2003). Disassembly sequencing: A survey. *International Journal of Production Research, 41*(16), 3721–3759.

Lambert, A. J. D., & Gupta, S. M. (2005). *Disassembly modeling for assembly, maintenance, reuse and recycling*. London: CRC press.

Lambert, A. J. (2007). Optimizing disassembly processes subjected to sequence-dependent cost. *Computers & Operations Research, 34*(2), 536–551.

McGovern, S. M., & Gupta, S. M. (2006). Ant colony optimization for disassembly sequencing with multiple objectives. *The International Journal of Advanced Manufacturing Technology, 30*(5), 481–496.

Rardin, R. L. (1998). *Optimization in operations research*. London: Prentice-Hall.

Rickli, J. L., & Camelio, J. A. (2013). Multi-objective partial disassembly optimization based on sequence feasibility. *Journal of Manufacturing Systems, 32*(1), 281–293.

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

Srinivasan, H., Figueroa, R., & Gadh, R. (1999). Selective disassembly for virtual prototyping as applied to de-manufacturing. *Robotic Computer-Integrated Manufacturing, 15*, 231–245.

Wang, J., Liu, J., Li, S., & Zhong, Y. (2003). Intelligent selective disassembly using the ant colony algorithm. *AI EDAM: Artificial Intelligence for Engineering Design, Analysis and Manufacturing, 17*(04), 325–333.

Zhong, L., Youchao, S., Gabriel, O. E., & Haiqiao, W. (2011). Disassembly sequence planning for maintenance based on metaheuristic method. *Aircraft Engineering and Aerospace Technology, 83*(3), 138–145.