Int J Adv Manuf Technol (2006) 30: 497–506
DOI 10.1007/s00170-005-0041-x

ORIGINAL ARTICLE

Elif Kongar · Surendra M. Gupta

Disassembly sequencing using genetic algorithm

Received: 1 July 2004 / Accepted: 17 March 2005 / Published online: 17 December 2005

Abstract At the end-of-life (EOL) of a product, there are several options available for its processing including reuse, remanufacturing, recycling and disposing. In almost all cases, a certain level of disassembly may be necessary. Thus, finding an optimal (or near optimal) disassembly sequence is crucial to increasing the efficiency of the process. Disassembly operations are labor intensive, can be costly, have unique characteristics and cannot be considered as the reverse of assembly operations. Since the complexity of determining the best disassembly sequence increases with increase in the number of parts of the product, it is extremely crucial that an efficient methodology for disassembly sequencing be developed. In this paper, we present a genetic algorithm for disassembly sequencing of EOL products. A case example is considered to demonstrate the functionality of the algorithm.

Keywords Disassembly · Disassembly sequencing · Genetic algorithm · Product recovery

1 Introduction

In recent years, the growing amount of waste generated at the end-of-life (EOL) of products has become a severe problem in many countries. Many governments are now holding the manufacturers responsible to take-back their products at the end of their lives. In response, the manufacturers are seeking solutions to address the potential accumulation of large inventories. This inventory may consist of outdated and/or non-functioning products. As far as the electronic products are concerned, which is the focus of this paper, the rapid pace of technological development has the potential to render the products obsolete after a short time, even though most of them may still be in excellent working conditions. The challenge is to process these products in an environmentally benign and cost effective manner. There are various ways to accomplish this task under the general umbrella of end-of-life (EOL) processing. Some of the options for EOL processing include reuse, recycle, and storage for future use or disposal. Even though the disposal option is not desirable, there are times when it is the only option available. In such cases, the method of disposal chosen should be least harmful to the environment.

In the majority of EOL processing, a certain level of disassembly may be necessary. Even in the disposal option, the hazardous contents are separated from the product and carefully processed before the residual product is disposed of. *Disassembly* is the process of systematic removal of desirable constituents (components and/or materials) from the original assembly so that there is no impairment to any useful constituent. Disassembly can be *partial* (product not fully disassembled) or *complete* (product fully disassembled) and may use a methodology that is *destructive* (focusing on material rather than component recovery) or *non-destructive* (focusing on component rather than material recovery). In this paper, we consider the case of complete disassembly where the components are retrieved by either destructive or non-destructive methodology.

Many products are made up of a large number of components. The optimal disassembly path to retrieve the components can theoretically be obtained using exhaustive search algorithms. However, because of the combinatorial nature of the problem, as the number of components in a product grow, the number of possible disassembly combinations grows dramatically, making the path selection

E. Kongar
Department of Industrial Engineering,
Yildiz Technical University,
Istanbul, Turkey

S. M. Gupta (✉)
Laboratory for Responsible Manufacturing,
334 SN, Department of Mechanical
and Industrial Engineering,
Northeastern University,
360 Huntington Avenue,
Boston, MA 02115, USA
e-mail: gupta@neu.edu
Tel.: +1-617-3734846
Fax: +1-617-3732921

practically prohibitive from the point of view of cost and time required. For this reason, heuristic methods are often employed to find near-optimal or optimal solutions. These approaches can frequently provide satisfactory results much faster and at low costs. In this paper, we present a genetic algorithm for disassembly sequencing in the presence of constraints and precedence relationships.

This paper is organized as follows. A brief literature review is provided in Section 2. Section 3 presents an overview of a genetic algorithm in connection with disassembly sequencing. A numerical example is presented in Section 4 and the conclusions are presented in Section 5.

2 Literature review

Various researchers have studied disassembly. It is one of the primary elements for parts and product recovery. However, disassembly problems are not simple as they are likely to be *NP*-complete. Gupta and Taleb [1], Taleb and Gupta [2] and Taleb et al. [3] proposed algorithms for scheduling the disassembly of discrete and well-defined product structures. Veerakamolmal et al. [4] presented a methodology for disassembly process planning. Veerakamolmal and Gupta [5] proposed a method that provides a solution for component recovery planning. Veerakamolmal and Gupta [6] developed a design for disassembly index (DfDI) for measuring the products' design efficiencies from the EOL point of view. Kuo [7] analyzed the cost of disassembly in electromechanical products. Torres et al. [8] reported a study for non-destructive automatic disassembly of personal computers. Moore et al. [9] used Petri-nets to study products with complex AND/OR relationships. Gungor and Gupta [10] presented an approach to generate a disassembly precedence matrix and disassembly sequence plan for products with limited number of parts that require complete disassembly. Recently, Lambert and Gupta [11] addressed the problem of demand driven disassembly using a tree network model while Kongar and Gupta [12] used goal programming to address such problems. For more information on disassembly, see Brennan et al. [13], Gupta and McLean [14], Moyer and Gupta [15], Gungor and Gupta [16], Tang et al. [17], Lee et al. [18] and Lambert [19]. A recent book by Lambert and Gupta [20] is also helpful in understanding the general area of disassembly.

In recent years, genetic algorithm (GA) has been gaining popularity for solving combinatorial and *NP*-complete problems. As mentioned before, EOL processing often necessitates a certain level of disassembly, which is a relatively expensive process due to its labor-intensive nature. Hence, finding an optimal or near optimal disassembly sequence is desirable. GA is a heuristic technique that can provide a quick and cost effective solution to a problem that would otherwise take an excessive amount of time to render it practical. The price one has to pay is in terms of the quality of solution one gets. Even though by using GA an optimal solution cannot always be guaranteed, a reasonable (and in many cases optimal) solution is often obtained. Thus, GA offers a good compromise for a large class of problems including disassembly sequencing.

Valenzuela-Rendón and Uresti-Charre [21] proposed a GA for multi-objective optimization. They calculated the fitness of each entity in the population incrementally based on the degree to which it is dominated or how close it is to other entities. The authors used a sharing function to measure the closeness of entities. A comparison is also provided with previous studies on three multi-objective optimization problems of growing difficulty. The behavior of each algorithm is then analyzed with regard to the visited search space, the quality of the final population attained and the percentage of non-dominated entities in the population through time. The authors concluded that GA had more stable and reliable time response.

Keung et al. [22] also applied a multi-objective GA approach to a tool selection model. In their paper, the overall objective of the model was to minimize the processing time. Loughlin and Ranjithan [23] proposed a GA method, a so-called neighborhood constraint method, and concluded that the GA performed better in multi-objective problems compared to single objective problems.

Lazzerini and Marcelloni [24] used GA in scheduling assembly processes. They employed a modified partially matched crossover (PMX) and mutation operations to obtain the near optimal sequence. The precedence relationships were not considered in the model.

Precedence relationships are one of the factors that add to complications in sequencing problems. The conventional search algorithms generally use combinatorial search techniques and then augment them with precedence relationships. Sanderson et al. [25] considered precedence relationships in assembly sequence planning in such a manner. Similarly, regular GAs are generally not suitable for the systems where precedence relationships and constraints are involved. For example, Seo et al. [26] recently proposed a genetic algorithm for generating optimal disassembly sequences considering both economical and environmental factors. Their algorithm aimed to obtain the optimal disassembly sequence. However, their search can lead to infeasible strings during the crossover and mutation operations. The authors addressed this situation by penalizing the string with the hope that it would be eliminated during latter generations. Therein lies a weakness in their algorithm. Bierwirth et al. [27], and Bierwirth and Mattfeld [28] proposed a methodology to overcome this problem by introducing the precedence preservative crossover (PPX) technique for scheduling problems. The methodology preserves the precedence relationships during the crossover function of GA. The method guarantees feasible results at each of the steps. This approach is also employed in the GA application to disassembly sequencing of this paper. Even though the two methods are similar, the model proposed in this paper provides a genetic algorithm approach for multiple criteria modeling whereas Bierwirth et al. [27], and Bierwirth and Mattfeld [28] employed the approach to a single objective job shop scheduling problem.

3 Overview of genetic algorithm

Genetic algorithm got its inspiration from Darwin's theory of evolution and can be briefly described as follows. The algorithm starts with a set of randomly selected potential solutions called the *population*. Each member of the population is encoded as an artificial *chromosome*, represented by a combination of numbers, alphabets and/or other characters, which contain information about the solution mapping. Each chromosome is assigned a score based on a predefined *fitness function*. A new population of chromosomes is iteratively created in the hope of finding a chromosome with a better score. At each step of creation, a *mutation* may occur in a chromosome, or two chromosomes may mate to produce a child; this process is known as *crossover*. (Note that it is possible to preserve feasibility in each solution of the new generation by carefully carrying information of feasibility from one generation to the next). The selection of parent chromosomes is biased towards fitter members of the population. The process is iterated until some predetermined objectives are achieved.

3.1 Nomenclature

Notation	Definition
C	Conversion constant (s)
chl	Length of chromosome
ch	Index for chromosome
CT	Total penalty for direction change (s)
$ct_{j,seq}$	Penalty for direction change for disassembling component j in sequence seq (0: if direction change is not required, 1: if 90° change is required, 2: if 180° change is required) (s)
$de_{j,seq}$	Type of demand for component j in sequence seq (s) (0: if not demanded, 1: if demanded for reuse, 2: if demanded for recycling)
$dt_{j,seq}$	Time required to disassemble component j in sequence seq (s)
DT	Total basic disassembly time till the *seq*th sequence (s)
$F(ch,gn)$	Fitness value for chromosome ch in generation gn (s)
gn	Index for generation
j	Index for component
$ma_{j,seq}$	Material type of component j in sequence seq (A: aluminum, P: plastic, S: steel)
$MS(ch,gen)$	Total makespan of chromosome ch in generation gen (s)
MT	Total penalty for disassembly method change (s)
$mt_{j,seq}$	Penalty for disassembly method change for disassembling component j in sequence seq (0: if method change is not required 1: if method change is required) (s)
n	Number of components in the EOL product (unit)
ncr	Number of chromosomes in the population (unit)
pop	Index for population
rnd	Random number (rnd=1,...,9)
seq	Index for disassembly sequence (seq=1,...,9)
T_{seq}	Cumulative disassembly time after component j in sequence seq is disassembled (s)

3.2 Elements of genetic algorithm for disassembly sequencing

In the following subsections we describe the elements and development of genetic algorithm with the help of an example as it pertains to disassembly sequencing. The structure of the example product is given in Fig. 1. It consists of ten components indexed by integers from 0 to 9. Therefore, $j\in\{0,1, .. , n-1\}$. Each component has to be disassembled in a 3D space in one of the six possible directions, viz., x, y, z, $-x$, $-y$, or $-z$. The methodology

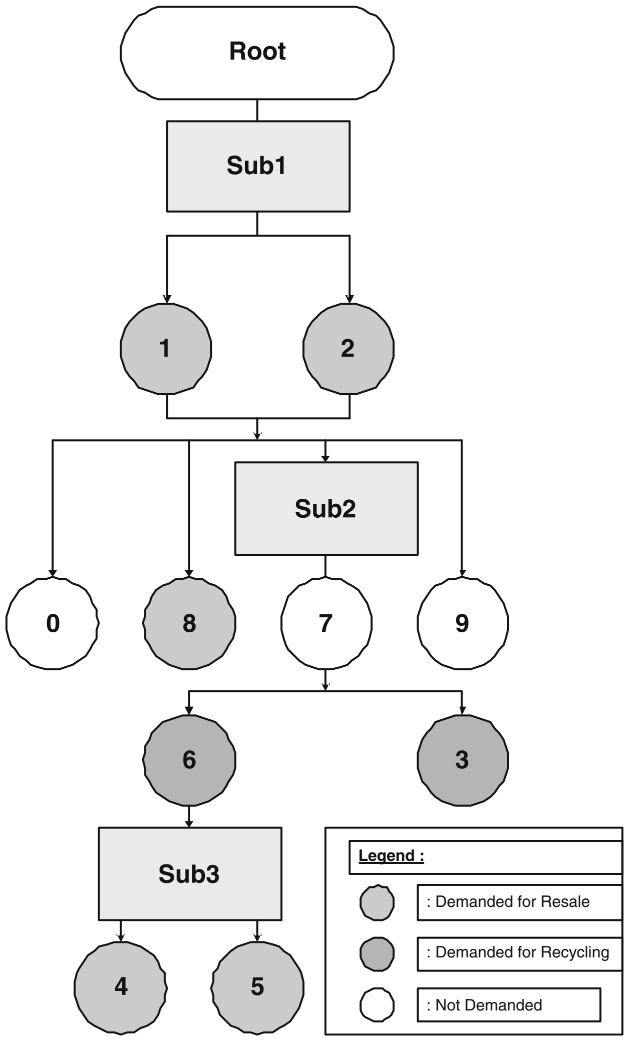

Fig. 1 Original product structure of the EOL product

used for the disassembly of each component may be destructive (D) or non-destructive (N). A component may or may not be demanded. If it is not demanded, it is represented by 0. If it is demanded, it may be demanded for reuse (represented by 1) or recycling (represented by 2). A component is made up of one of three types of materials, viz., aluminum (A), plastic (P) or steel (S). The specific data for each component for the product given in Fig. 1 is presented in Table 1. In addition, the precedence relationships are given as follows: component 1 **or** 2 must be disassembled prior to any other component; component 6 must be disassembled prior to components 4 **and** 5; and component 7 must be disassembled prior to components 6 **and** 3.

3.2.1 Chromosomes

In every GA problem, the solution and parameters must be coded into chromosomes before they can be processed. As mentioned earlier, a chromosome is represented by a combination of numbers, alphabets and/or other characters. The development of the representation is actually a non-trivial endeavor, as it must capture the key features of the problem in such a way that the desired characteristics are propagated while the undesirable ones are suppressed as the solution progresses from one generation to the next.

In order to capture the five variables (presented in Table 1) in a chromosome, we codify them in the form of a string consisting of five ordered sections of equal length, representing the disassembly sequence, the disassembly direction, the disassembly method, the demand type of each component and the material type of each component. An example of the chromosome structure is as follows:

Sequence	Direction	Method	Demand	Material
28**7**1063954,	$+x-z-\boldsymbol{x}+x+y+y+z-y-z-z$,	NN**D**NDD DDNN,	11**0**1022011,	SP**P**SASS PPP

This chromosome can be interpreted as follows. The first ten characters represent the first section of the chromosome and depict the disassembly sequence order of each component. The next ten characters represent the second section of the chromosome and depict the disassembly directions of the ten components in the first section respectively. The third, fourth and the fifth sections can also be similarly interpreted. In order to see how to extract information from the above chromosome, consider component 7 as an example. Component 7 is to be disassembled after component 8 and prior to component 1, requires a disassembly in the "$-x$" direction, is subjected to destructive disassembly (represented by D), is not demanded (represented by 0) and is made of plastic material (represented by P).

Table 1 Input data for the genetic algorithm example

Variable	Value									
j	0	1	2	3	4	5	6	7	8	9
Direction	$+y$	$+x$	$+x$	$+z$	$-z$	$-z$	$-y$	$-x$	$-z$	$-y$
Method	D	N	N	D	N	N	D	D	N	D
Demand	0	1	1	2	1	1	2	0	1	0
Material	A	S	S	S	P	P	S	P	P	P

3.2.2 Initial population

The initial population consists of *ncr* chromosomes. A repetitive random selection is performed to generate these chromosomes. This is done such that all the precedence relationships and any other constraints imposed by the product structure are satisfied. For the example given in Fig. 1, we chose $ncr=10$ and randomly create ten chromosomes (feasible solutions) to form the initial population using the initial data given in Table 1.

The random population with all the chromosomes is given in Table 2. As it can be observed from the table, all precedence relationships are preserved.

In order to improve the initial population, certain string manipulations are necessary. This is done by using genetic operators. There are various genetic operators for every genetic algorithm application, namely, crossover and mutation. These operators are explained in the following subsections. An additional operator, cloning, is described with the selection methodology.

3.2.3 Crossover

We employ the precedence preservative crossover (PPX) methodology for this operation. In this methodology, in addition to the two strings representing the chromosomes of the parents, two additional strings representing operators for crossover are added.

These operators pass on the precedence relationship based on the two parental permutations to two new offspring while making sure that no new precedence relationships are introduced. A vector of length *n* representing the number of operations involved in the problem—equal to the number of components in the EOL product—is randomly filled with elements of the set. This vector defines the order in which the operations are successively drawn from Parent_1 and Parent_2.

The algorithm starts by initializing an empty offspring. The leftmost operation in one of the two parents is selected in accordance with the order of parents given in the vector. After an operation is selected it is deleted in both parents. Finally, the selected operation is appended to the offspring. This step is repeated until both parents are empty and the offspring contains all operations involved.

As an example, consider two chromosomes, Parent_1 and Parent_2. The strings of these chromosomes are as follows:

Parent_1:
2871063954, $+x\ -z\ -x\ +x\ +y\ +y\ +z\ -y\ -z\ -z$, NNDNDD DDNN, 1101022011, SPPSASSPPP

Table 2 Initial population for the genetic algorithm example

No.	Chromosome
1	2,8,7,1,0,6,3,9,5,4,+*x*,−*z*,−*x*,+*x*,+*y*,+*y*,+*z*,−*y*,−*z*,−*z*,N,N,D,N,D,D,D,D,N,N,1,1,0,1,0,2,2,0,1,1,S,P,P,S,A,S,S,P,P,P
2	1,2,7,6,3,4,0,8,5,9,+*x*,+*x*,−*x*,+*y*,+*z*,−*z*,+*y*,−*z*,−*z*,−*y*,N,N,D,D,D,N,D,N,N,D,1,1,0,2,2,1,0,1,1,0,S,S,P,S,S,P,A,P,P,P
3	1,7,0,8,2,6,3,9,5,4,+*x*,−*x*,+*y*,−*z*,+*x*,+*y*,+*z*,−*y*,−*z*,−*z*,N,D,D,N,N,D,D,D,N,N,1,0,0,1,1,2,2,0,1,1,S,P,A,P,S,S,S,P,P,P
4	1,7,8,0,6,3,2,9,5,4,+*x*,−*x*,−*z*,+*y*,+*y*,+*z*,+*x*,−*y*,−*z*,−*z*,N,D,N,D,D,D,N,D,N,N,1,0,1,0,2,2,1,0,1,1,S,P,P,A,S,S,S,P,P,P
5	1,8,9,7,2,0,6,5,3,4,+*x*,−*z*,−*y*,−*x*,+*x*,+*y*,+*y*,−*z*,+*z*,−*z*,N,N,D,D,N,D,D,N,D,N,1,1,0,0,1,0,2,1,2,1,S,P,P,P,S,A,S,P,S,P
6	1,2,8,7,9,0,6,3,5,4,+*x*,+*x*,−*z*,−*x*,−*y*,+*y*,+*y*,+*z*,−*z*,−*z*,N,N,N,D,D,D,D,D,N,N,1,1,1,0,0,0,2,2,1,1,S,S,P,P,P,A,S,S,P,P
7	1,0,2,7,3,6,4,5,8,9,+*x*,+*y*,+*x*,−*x*,+*z*,+*y*,−*z*,−*z*,−*z*,−*y*,N,D,N,D,D,D,N,N,N,D,1,0,1,0,2,2,1,1,1,0,S,A,S,P,S,S,P,P,P,P
8	1,0,2,7,3,6,5,9,4,8,+*x*,+*y*,+*x*,−*x*,+*z*,+*y*,−*z*,−*y*,−*z*,−*z*,N,D,N,D,D,D,N,D,N,N,1,0,1,0,2,2,1,0,1,1,S,A,S,P,S,S,P,P,P,P
9	1,7,8,6,3,9,5,4,0,2,+*x*,−*x*,−*z*,+*y*,+*z*,−*y*,−*z*,−*z*,+*y*,+*x*,N,D,N,D,D,D,N,N,D,N,1,0,1,2,2,0,1,1,0,1,S,P,P,S,S,P,P,P,A,S
10	1,8,9,7,6,5,0,3,4,2,+*x*,−*z*,−*y*,−*x*,+*y*,−*z*,+*y*,+*z*,−*z*,+*x*,N,N,D,D,D,N,D,D,N,N,1,1,0,0,2,1,0,2,1,1,S,P,P,P,S,P,A,S,P,S

Parent$_2$:
1276340859, $+x +x -x +y +z -z +y -z -z -y$, NNDDDN DNND, 1102210110, SSPSSPAPPP

Since the offspring depends on only the first part of the parents' chromosomes, we can simplify the two strings as follows:

Parent$_1$: 2 8 7 1 0 6 3 9 5 4
Parent$_2$: 1 2 7 6 3 4 0 8 5 9

Next, the mask arrays for the two children are created. The masks consist of numbers 1 and 2 representing the parent number from which the gene is selected.

Assume that the two random masks for Child$_1$ and Child$_2$ are as follows:

Mask for Child$_1$: 2 2 2 1 1 2 2 2 1 2
Mask for Child$_2$: 2 1 1 2 1 2 2 1 2 2

Then, the first part of the chromosomes of Child$_1$ and Child$_2$ become:

Child$_1$: 1 2 7 8 0 6 3 4 9 5
Child$_2$: 1 2 8 7 0 6 3 9 4 5

Hence, the chromosomes of Child$_1$ and Child$_2$ can be written as:

Child$_1$:
1278063495, $+x +x -x -z +y +y +z -z -y -z$, NNDNDD DNDD, 1101022101, SSPPASSPPP
Child$_2$:
1287063945, $+x +x -z -z +y +y +z -y -z -z$, NNNDDD DDNN, 1110022011, SSPPASSPPP

3.2.4 Mutation

After the crossover operation the population is subjected to a mutation operation. The chromosomes mutate with a given probability. If the probability holds, the mutation operator selects a random number of genes (*rnd*=1, ... ,9), and exchanges them in such a way that the same precedence relationships are preserved. Otherwise the population remains unchanged and is copied to the next generation. The mutation operator proposed in this paper exchanges components 1 and 2. As an example, consider the mutation operation given in Table 3.

As it can be seen from the new population, only the first four (randomly selected number) chromosomes are selected for mutation. The rest of the strings remain the same. This is done to preserve some of the strong solutions in the current population.

3.2.5 Fitness evaluation

Whole fitness function is dependent on the increment in disassembly time. There are three factors, which add up to the disassembly time of a component. The first one is basic disassembly time for component *j* in sequence *seq* ($dt_{j,seq}$). In this paper $dt_{j,seq}$ values (in seconds) are given as:

j	0	1	2	3	4	5	6	7	8	9
$dt_{j,seq}$	2	3	3	2	3	4	2	1	3	2

Table 3 Numerical example for the proposed mutation operator

Chromosome no.	Before mutation	After mutation
1	**2** 8 7 **1** 0 6 3 9 5 4	**1** 8 7 **2** 0 6 3 9 5 4
2	**1 2** 7 6 3 4 0 8 5 9	**2 1** 7 6 3 4 0 8 5 9
3	**1** 7 0 8 **2** 6 3 9 5 4	**2** 7 0 8 **1** 6 3 9 5 4
4	**1** 7 8 0 6 3 **2** 9 5 4	**2** 7 8 0 6 3 **1** 9 5 4
5	1 8 9 7 2 0 6 5 3 4	1 8 9 7 2 0 6 5 3 4
6	1 2 8 7 9 0 6 3 5 4	1 2 8 7 9 0 6 3 5 4
7	1 0 2 7 3 6 4 5 8 9	1 0 2 7 3 6 4 5 8 9
8	1 0 2 7 3 6 5 9 4 8	1 0 2 7 3 6 5 9 4 8
9	1 7 8 6 3 9 5 4 0 2	1 7 8 6 3 9 5 4 0 2
10	1 8 9 7 6 5 0 3 4 2	1 8 9 7 6 5 0 3 4 2

The second function in fitness evaluation is the penalty (in seconds) for each direction change ($ct_{j,seq}$) for component j in sequence seq, where

$$ct_{j,seq} = \begin{cases} 0, & \text{If no direction change is required,} \quad (\text{e.g., } +x \text{ to } +x) \\ 1, & \text{If } 90^{o} \text{ change is required,} \quad (\text{e.g., } +x \text{ to } +y) \\ 2, & \text{If } 180^{o} \text{ change is required,} \quad (\text{e.g., } +x \text{ to } -x) \end{cases}$$

The last criterion in fitness function is the penalty for disassembly method change ($mt_{j,seq}$). For each disassembly method change, the sequence is penalized by 1 s. Hence, mathematically,

$$mt_{j,seq} = \begin{cases} 0, & \text{If no method change is required,} \quad (\text{e.g., N to N}) \\ 1, & \text{If method change is required,} \quad (\text{e.g., N to D}) \end{cases}$$

Let T_{seq} denote the cumulative disassembly time after the disassembly operation in sequence seq is completed for component j. Mathematically T_{seq} can be expressed as follows:

$$\begin{aligned} T_{seq} &= T_{seq-1} + dt_{j,seq} + ct_{j,seq} + mt_{j,seq}, \text{ for } seq = 0, \ldots, n-2 \\ T_{seq} &= T_{seq-1} + dt_{j,seq}, \text{ for } seq = n-1 \end{aligned} \quad (1)$$

Let *MS(ch,gn)* denote the total makespan for chromosome *ch* in generation *gn*. Hence, total time to disassemble all the components can be calculated as follows:

$$MS = \sum_{seq=0}^{n-1} dt_{j,seq} + \sum_{seq=0}^{n-2} ct_{j,seq} + \sum_{seq=0}^{n-2} mt_{j,seq}, \quad \forall j, j = 0, \ldots, n-1 \quad (2)$$

Equations (1) and (2) are true unless the proposed algorithm defines a pair that will not be detached from each other. A "pair" represents two consecutive components, which are made of the same material, and both are demanded for recycling. The algorithm searches the pair in each consecutive sequence. If there is a successful pair, the disassembly time, direction and disassembly method change penalties are assigned to zero, implying these components will not be separated from each other and will be disassembled as a combination from the EOL product. This can be expressed mathematically:

$$\textbf{\textit{If}}\ (de_{j,seq} = de_{j,seq+1} = 2) \text{ and } (ma_{j,seq} = ma_{j,seq+1}) \Rightarrow T_{seq} = T_{seq-1}, \forall j, seq \quad (3)$$

$$\textbf{\textit{Else}} \quad \Rightarrow \text{Eq.}(1) \quad (4)$$

where j=0,1,..., $n-1$, seq=0,1,..., $n-1$, and n is the number of components in the product structure.

In this proposed GA model, the objective is to minimize the total makespan (*MS*) by minimizing (i) the number of direction changes, (ii) the number of disassembly method changes, and (iii) by combining the identical-material components together, eliminating unnecessary disassembly operations. But, as with all GAs, the model is going to select the chromosome, which will provide higher fitness value. Hence, the fitness function has to be converted to a maximization function *F(ch,gn)*, given below:

$$F(ch, gn) = C - MS(ch, gn) \quad (5)$$

where, *F(ch,gn)* is the fitness value of the *ch*th chromosome in the *gn*tth generation. *C* is a conversion constant which will convert the "lower is better - minimization" function to a "higher is better - maximization" model. So, *C* should be chosen so that it will always satisfy the following condition:

$$C > MS(ch, gn) \tag{6}$$

3.2.6 Illustration of fitness evaluation

To provide better understanding, let us consider the following string example.

2817069354, *+x−z+x−x+y+y−y+z−z−z*, NNNDDDDDNN, 1110020211, SPSPASPSPP

Calculation of the fitness of this sequence is explained below:

3.2.6.1 Penalty for the direction change (CT) Necessary data for calculation of the penalty for direction change data is given as follows:

Sequence (*seq*):	0	1	2	3	4	5	6	7	8	9
Component (*j*):	2	8	1	7	0	6	9	3	5	4
Partial string:	+*x*	−*z*	+x	−*x*	+y	+y	−*y*	+z	−*z*	−*z*
Change (degrees):	90	90	180	90	0	180	90	180	0	–
$ct_{j,seq}$ (s):	1	1	2	1	0	2	1	2	0	–

Hence, penalty for direction change can be calculated as:

$$\begin{aligned} CT &= \sum\nolimits_{seq=0}^{n-2} ct_{j,seq}, \forall j \\ &= 1+1+2+1+0+2+1+2+0 \\ &= 10\,\mathrm{s} \end{aligned}$$

Note that there are $(n-1)$ penalties for n components since penalty denotes the change in direction and does not include the gene.

3.2.6.2 Penalty for the disassembly method change (MT) The penalty for the disassembly method change data is given as follows:

Sequence (*seq*):	0	1	2	3	4	5	6	7	8	9
Component (*j*):	2	8	1	7	0	6	9	3	5	4
Partial string:	N	N	N	D	D	D	D	D	N	N
Change (1:yes; 0 :no):	0	0	1	0	0	0	0	1	0	–
$mt_{j,seq}$ (s):	0	0	1	0	0	0	0	1	0	–

Hence, penalty for disassembly method change can be calculated as:

$$\begin{aligned} MT &= \sum\nolimits_{seq=0}^{n-2} mt_{j,seq}, \forall j \\ &= 0+0+1+0+0+0+0+1+0 \\ &= 2\,\mathrm{s} \end{aligned}$$

Similar to the calculation of direction change, the component in the last sequence is not taken into account.

3.2.6.3 Total disassembly time (DT) Since there is no recycling pair in this sequence, all the components are subject to disassembly. Hence, the disassembly time data can be can be given as follows:

Sequence (*seq*):	0	1	2	3	4	5	6	7	8	9
Component (*j*):	2	8	1	7	0	6	9	3	5	4
$dt_{j,seq}$ (s):	3	3	3	1	2	2	2	2	4	3

Table 4 System parameters for the GA numerical example

Parameter	Explanation
Initial population	Random and feasible
Population size	10
Length of the chromosome (*chl*)	*n* * 5 (50)
Max. number of generations	100
Crossover operator	Precedence preservative crossover (PPX)
Mutation operator	Applies to a random number (*rnd*) of chromosomes ($0<rnd<n-1$) and replaces all component 1s with component 2s in the corresponding sequences
Crossover probability	0.60
Mutation probability	0.005
Selection procedure	Half of the best-fit chromosomes are selected for the next generation
Regeneration procedure	Selected chromosomes are cloned to keep the population size constant
Fitness parameters	Basic disassembly time, penalty for direction change, penalty for method change
Termination conditions	When the maximum number of generations are exceeded or no further improvement is obtained

Table 5 Initial population for the numerical example

ch	Chromosome String	*F*(*ch,gn*)
0	2 8 7 1 0 6 3 9 5 4, +*x* −*z* −*x* +*x* +*y* +*y* +*z* −*y* −*z* −*z*, N N D N D D D D N N, 1 1 0 1 0 2 2 0 1 1, S P P S A S S P P P	66
1	1 2 7 6 3 4 0 8 5 9, +*x* +*x* −*x* +*y* +*z* −*z* +*y* −*z* −*z* −*y*, N N D D D N D N N D, 1 1 0 2 2 1 0 1 1 0, S S P S S P A P P P	64
2	1 7 0 8 2 6 3 9 5 4, +*x* −*x* +*y* −*z* +*x* +*y* +*z* −*y* −*z* −*z*, N D D N N D D D N N, 1 0 0 1 1 2 2 0 1 1, S P A P S S S P P P	65
3	1 7 8 0 6 3 2 9 5 4, +*x* −*x* −*z* +*y* +*y* +*z* +*x* −*y* −*z* −*z*, N D N D D D N D N N, 1 0 1 0 2 2 1 0 1 1, S P P A S S S P P P	64
4	1 8 9 7 2 0 6 5 3 4, +*x* −*z* −*y* −*x* +*x* +*y* +*y* −*z* +*z* −*z*, N N D D N D D N D N, 1 1 0 0 1 0 2 1 2 1, S P P P S A S P S P	58
5	1 2 8 7 9 0 6 3 5 4, +*x* +*x* −*z* −*x* −*y* +*y* +*y* +*z* −*z* −*z*, N N N D D D D D N N, 1 1 1 0 0 0 2 2 1 1, S S P P P A S S P P	68
6	1 0 2 7 3 6 4 5 8 9, +*x* +*y* +*x* −*x* +*z* +*y* −*z* −*z* −*z* −*y*, N D N D D D N N N D, 1 0 1 0 2 2 1 1 1 0, S A S P S S P P P P	65
7	1 0 2 7 3 6 5 9 4 8, +*x* +*y* +*x* −*x* +*z* +*y* −*z* −*y* −*z* −*z*, N D N D D D N D N N 1 0 1 0 2 2 1 0 1 1, S A S P S S P P P P	63
8	1 7 8 6 3 9 5 4 0 2, +*x* −*x* −*z* +*y* +*z* −*y* −*z* −*z* +*y* +*x*, N D N D D D N N D N, 1 0 1 2 2 0 1 1 0 1, S P P S S P P P A S	63
9	1 8 9 7 6 5 0 3 4 2, +*x* −*z* −*y* −*x* +*y* −*z* +*y* +*z* −*z* +*x*, N N D D D N D D N N, 1 1 0 0 2 1 0 2 1 1, S P P P S P A S P S	61

Thus, disassembly time =

$$DT = \sum_{seq=0}^{n-1} dt_{j,seq}, \forall j$$
$$= 3 + 3 + 3 + 1 + 2 + 2 + 2 + 2 + 4 + 3$$
$$= 25\,\mathrm{s}$$

Since there is no pair in that sequence, we can calculate the total disassembly time for this sequence as in Eq. (2). Therefore,

$$T_9 = CT + MT + DT$$
$$T_9 = 10 + 2 + 25 = 37\,\mathrm{s}$$

Since this gives the time after the final disassembly operations, from Eqs. (1) and (2), total makespan becomes *MS*=37 s. Let *C*=100. Then, the fitness of the sequence can be calculated using Eq. (5) as $100 - 37 = 63\,\mathrm{s}$.

3.2.7 Selection and regeneration procedure

After every generation, the chromosomes in the current population are sorted according to their fitness values. The first half of the sorted population is selected and cloned. This way, a new population is generated eliminating the weak chromosomes. This method seeks to improve the population while allowing the better chromosome generation with further string manipulations such as crossover and mutation.

3.2.8 Termination

The execution of GA terminates if one of the following two conditions is met: the number of generations reaches up to a maximum value (100 in our example), or the ratio of the average fitness value of the new population to the average fitness value of the old population has a value less than or equal to a certain number (1.0005 in our example), i.e., the population is not improving any further.

3.2.9 Genetic algorithm model assumptions

In this model, we assumed the following:

- The direction of each disassembly operation is independent of the adjacent components in the sequence.
- Neither non-destructive nor destructive disassembly destroys the adjacent components during the operation.
- Complete disassembly is required for the EOL product; hence, each component will be separated from each other unless they are defined as a *unit*.

Table 6 Final population for the numerical example

ch	*F(ch,gn)*	*F(ch,gn)*
0	2 1 0 7 3 6 9 4 5 8, +*x* +*x* +*y* −*x* +*z* −*y* −*y* −*z* −*z* −*z*, N N D D D D D N N N, 1 1 0 0 2 2 0 1 1 1, S S A P S S P P P	71
1	2 1 0 7 3 6 9 4 5 8, +*x* +*x* +*y* −*x* +*z* −*y* −*y* −*z* −*z* −*z*, N N D D D D D N N N, 1 1 0 0 2 2 0 1 1 1, S S A P S S P P P	71
2	2 1 0 7 3 6 9 4 5 8, +*x* +*x* +*y* −*x* +*z* −*y* −*y* −*z* −*z* −*z*, N N D D D D D N N N, 1 1 0 0 2 2 0 1 1 1, S S A P S S P P P	71
3	2 1 0 7 3 6 9 4 5 8, +*x* +*x* +*y* −*x* +*z* −*y* −*y* −*z* −*z* −*z*, N N D D D D D N N N, 1 1 0 0 2 2 0 1 1 1, S S A P S S P P P	71
4	2 1 0 7 3 6 9 4 5 8, +*x* +*x* +*y* −*x* +*z* −*y* −*y* −*z* −*z* −*z*, N N D D D D D N N N, 1 1 0 0 2 2 0 1 1 1, S S A P S S P P P	71
5	2 1 0 7 3 6 9 4 5 8, +*x* +*x* +*y* −*x* +*z* −*y* −*y* −*z* −*z* −*z*, N N D D D D D N N N, 1 1 0 0 2 2 0 1 1 1, S S A P S S P P P	71
6	2 1 0 7 3 6 9 4 5 8, +*x* +*x* +*y* −*x* +*z* −*y* −*y* −*z* −*z* −*z*, N N D D D D D N N N, 1 1 0 0 2 2 0 1 1 1, S S A P S S P P P	71
7	2 1 0 7 3 6 9 4 5 8, +*x* +*x* +*y* −*x* +*z* −*y* −*y* −*z* −*z* −*z*, N N D D D D D N N N, 1 1 0 0 2 2 0 1 1 1, S S A P S S P P P	71
8	2 1 0 7 3 6 9 4 5 8, +*x* +*x* +*y* −*x* +*z* −*y* −*y* −*z* −*z* −*z*, N N D D D D D N N N, 1 1 0 0 2 2 0 1 1 1, S S A P S S P P P	71
9	2 1 0 7 3 6 9 4 5 8, +*x* +*x* +*y* −*x* +*z* −*y* −*y* −*z* −*z* −*z*, N N D D D D D N N N, 1 1 0 0 2 2 0 1 1 1, S S A P S S P P P	71

Table 7 Optimal exhaustive search results

ch	Sequence	*F(ch,gn)*
0	1 2 0 7 3 6 9 4 5 8	71
1	1 2 0 7 3 6 9 4 8 5	71
2	1 2 0 7 3 6 9 5 4 8	71
3	1 2 0 7 3 6 9 5 8 4	71
4	1 2 0 7 3 6 9 8 4 5	71
5	1 2 0 7 3 6 9 8 5 4	71
6	2 1 0 7 3 6 9 4 5 8	71
7	2 1 0 7 3 6 9 4 8 5	71
8	2 1 0 7 3 6 9 5 4 8	71
9	2 1 0 7 3 6 9 5 8 4	71

3.2.10 Genetic algorithm model steps

The steps of the proposed GA can be summarized as follows.

Step 1. Start with a population of *ncr* random individuals each with *chl*-length chromosomes.

Step 2. Calculate the fitness *F(ch,gn)* of each chromosome (*ch*) in the generation (*gn*).

Step 3. Sort the chromosomes in an ascending order according to their fitness.

Step 4. If the current population is not the initial population, select the first half of the sorted population and clone the selection to the generation.

Step 5. Based on fitness, select two best-fit chromosomes ($Parent_1$ and $Parent_2$) from the first half of the population.

Step 6. Randomly calculate the crossover probability. If probability holds perform precedence preservative crossover (PPX) operation, and generate the output of the crossover as the children ($Child_1$ and $Child_2$); if not, define parents ($Parent_1$ and $Parent_2$) as the children ($Child_1$ and $Child_2$).

Step 7. Place the two children into a new generation (*gn*+1).

Step 8. Return to Step 2 until the new generation contains *ncr* chromosomes. Then replace the old population with the new generation.

Step 9. Randomly calculate the mutation probability. If probability holds perform the mutation operation on a randomly selected number (*rnd*) of chromosomes starting from the first chromosome, generate the output of the mutation as a new population; if not, define the current population as the new population.

Step 10. If the termination condition is satisfied, STOP, else return to Step 1.

4 Numerical example

Consider the product in Fig. 1, with the initial data as given in Table 1. The crossover and mutation probabilities are assumed to be 0.60 and 0.005, respectively. The system parameters for this numerical example are listed in Table 4.

The initial population of ten chromosomes (*ncr*=10) is randomly generated and is as shown in Table 5.

After six generations (*gn*=6) the solution of our example is obtained in a negligible amount of execution time. The final population is as shown in Table 6.

As can be observed from the table, the sequence, (2 1 0 7 3 6 9 4 5 8, $+x +x +y -x +z -y -y -z -z -z$, N N D D D D D N N N, 1 1 0 0 2 2 0 1 1 1, S S A P S S P P P) is chosen to be the best chromosome. The fitness value is 71. Since the conversion constant is given as 100, the total makespan (*MS*) can be calculated as 29 s ($MS = 100 - 71$). It is important to note that the sequence is found in a few iterations even though it did not take place in the initial random population (See Table 5).

In order to provide a comparison, we solved the problem with an exhaustive search (ES) methodolody. ES obtained a total of twelve sequences with the minimum makespan of 29 s. The optimal sequences are given in Table 7. As it can be observed from the table, GA algorithm was able to obtain one of the optimal sequences in a few iterations. The codes for both GA and ES models were written in ANSI-C.

The problem was further explored for various combinations of population size, length of chromosomes, crossover probability and mutation probability, all of which gave satisfactory results.

5 Conclusions

In this paper, a genetic algorithm model was presented in order to determine the optimal disassembly sequence of a given product. The model provides quick and reliable input to the disassembly scheduling environments. As also stated by Keung et al. [22], GAs do not make unrealistic assumptions such as linearity, convexity and/or differentiability. This adds further importance to the proposed model and makes it even more desirable. For the example considered, the algorithm provided optimal disassembly sequence in a short execution time. The algorithm is practical, as it is easy to use, considers the precedence relationships and additional constraints in the product structure and is easily applicable to problems with multiple objectives.

References

1. Gupta SM, Taleb K (1994) Scheduling disassembly. Int J Prod Res 32(8):1857–1866
2. Taleb K, Gupta SM (1997) Disassembly of multiple product structures. Comput Ind Eng 32(4):949–961
3. Taleb KN, Gupta SM, Brennan L (1997) Disassembly of complex products with parts and materials commonality. Prod Plan Control 8(3):255–269
4. Veerakamolmal P, Gupta SM, McLean CR (1997) Disassembly process planning. 1st international conference on engineering design and automation, Bangkok, Thailand, pp 162–165

5. Veerakamolmal P, Gupta SM (1998) Optimal analysis of lot size balancing for multi-products selective disassembly. Int J Flex Automat Integr Manuf 6(3/4):245–269
6. Veerakamolmal P, Gupta SM (1999) Analysis of design efficiency for the disassembly of modular electronic products. J Electron Manuf 9(1):79–95
7. Kuo TC (2000) Disassembly sequence and cost analysis for electromechanical products. Robot Com-Int Manuf 16(1):43–54
8. Torres F, Gil P, Puente ST, Pomares J, Aracil R (2004) Automatic PC disassembly for component recovery. Int J Adv Manuf Technol 23(1/2):39–46
9. Moore KE, Gungor A, Gupta SM (2001) Petri net approach to disassembly process planning for products with complex AND/OR precedence relationships. Eur J Oper Res 135(2):428–449
10. Gungor A, Gupta SM (2001) Disassembly sequence plan generation using a branch-and-bound algorithm. Int J Prod Res 39(3):481–509
11. Lambert AJD, Gupta SM (2002) Demand-driven disassembly optimization for electronic products. J Electron Manuf 11 (2):121–135
12. Kongar E, Gupta SM (2002) A multi-criteria decision making approach for disassembly-to-order systems. J Electron Manuf 11(2):171–183
13. Brennan L, Gupta SM, Taleb KN (1994) Operations planning issues in an assembly/disassembly environment. Int J Oper Prod Man 14(9):57–67
14. Gupta SM, McLean CR (1996) Disassembly of products. Comput Ind Eng 31:225–228
15. Moyer L, Gupta SM (1997) Environmental concerns and recycling/disassembly efforts in the electronics industry. J Electron Manuf 7(1):1–22
16. Gungor A, Gupta SM (1999) Issues in environmentally conscious manufacturing and product recovery: a survey. Comput Ind Eng 36(4):811–853
17. Tang Y, Zhou M, Zussman E, Caudill R (2000) Disassembly modeling, planning, and application: a review. IEEE international conference on robotics and automation, San Francisco, California, pp 2197–2202
18. Lee D-H, Kang J-G, Xirouchakis P (2001) Disassembly planning and scheduling: review and further research. J Eng Manuf 215(B5):695–709
19. Lambert AJD (2003) Disassembly sequencing: a survey. Int J Prod Res 41(16):3721–3759
20. Lambert AJD, Gupta SM (2005) Disassembly modeling for assembly, maintenance, reuse, and recycling. CRC Press, Boca Raton, Florida
21. Valenzuela-Rendon M, Uresti-Charre E (1997) A non-generational genetic Algorithm for multiobjective optimization. 7th international conference on genetic algorithms, pp 658–665
22. Keung KW, Ip WH, Lee TC (2001) The solution of a multi-objective tool selection model using the GA approach. Int J Adv Manuf Technol 18:771–777
23. Loughlin DH, Ranjithan S (1997) The neighborhood constraint-method: a genetic algorithm-based multiobjective optimization technique. 7th international conference on genetic algorithms, pp 666–673
24. Lazzerini B, Marcelloni F (2000) A genetic algorithm for generating optimal assembly plans. Artif Intell Eng 14:319–329
25. Sanderson AC, Homem de Mello LS, Zhang H (1990) Assembly sequence planning. AI Magazine, Spring 1990, pp 62–81
26. Seo K-K, Park J-H, Jang D-S (2001) Optimal disassembly sequence using genetic algorithms considering economic and environmental aspects. Int J Adv Manuf Technol 18:371–380
27. Bierwirth C, Mattfeld DC, Kopfer H (1996) On permutation representations for scheduling problems. In: Voigt HM, Ebeling W, Rechenberg I, Schwefel H-P (eds) Parallel problem solving from nature-PPSN IV. Springer, Berlin Heidelberg New York, pp 310–318
28. Bierwirth C, Mattfeld DC (1999) Production scheduling and rescheduling with genetic algorithms. Evol Comput 7(1):1–18