Ashdin Publishing
International Journal of Swarm Intelligence and Evolutionary Computation
Vol. 1 (2012), Article ID Z110601, 7 pages
doi:10.4303/ijsiec/Z110601

Research Article

An Evolutionary Algorithm for Selective Disassembly of End-of-Life Products

Ahmed ElSayed,[1] Elif Kongar,[2] and Surendra M. Gupta[3]

[1]Department of Computer Science and Engineering, School of Engineering, University of Bridgeport, 221 University Avenue, 141 Technology Building, Bridgeport, CT 06604, USA
[2]Departments of Mechanical Engineering and Technology Management, School of Engineering, University of Bridgeport, 221 University Avenue, 141 Technology Building, Bridgeport, CT 06604, USA
[3]Laboratory for Responsible Manufacturing, 334 SN, Department of Mechanical and Industrial Engineering, Northeastern University, 360 Huntington Avenue, Boston, MA 02115, USA
Address correspondence to Elif Kongar, kongar@bridgeport.edu

Received 2 June 2011; Revised 28 September 2011; Accepted 2 October 2011

Abstract This paper addresses the problem of creating intelligent, green, and financially-beneficial disassembly sequences for end-of-life (EOL) electronic products. These complex EOL products contain a broad spectrum of materials including precious metals. Therefore, one would have to process these products to retrieve the value buried in them. EOL processing options include, reuse, remanufacturing, recycling or proper disposal. Each of this option requires a certain level of disassembly. Hence, obtaining an optimal or near optimal disassembly sequence is crucial to increasing the efficiency of EOL processing. Since the complexity of determining the best disassembly sequence increases as the number of parts in a product grows, an efficient methodology is required for disassembly sequencing. In this paper, we present an evolutionary algorithm for generating near-optimal and/or optimal sequences for selective disassembly of EOL products. A numerical example is provided to demonstrate the functionality of the algorithm.

Keywords selective disassembly sequencing; electronic waste; end-of-life; evolutionary computing; genetic algorithm

1 Introduction

Advanced technology products are regularly rendered technically obsolete within a few years of commercialization due to the rapid pace of technological enhancement. Thus, for example, electronic products are frequently discarded before their materials degrade. These complex end-of-life (EOL) products contain a broad spectrum of materials including precious metals such as silver and gold and valuable materials such as copper. Therefore, efficient recovery of materials in the electronic EOL products is essential not only for economic reasons but also for a sustainable environment. The practical lifetime of an electronic product depends primarily on the pace of superseding technological advancement that could make the otherwise fully-functioning product virtually obsolete.

However, the discarded product is likely to contain one or more usable component(s). The economically and environmentally sustainable option is to reuse these components in technically valid products. EOL processing options such as reuse, recycle and remanufacturing are effective ways to reclaim the materials and the components in electronic EOL products [5,8]. Regardless of the motivation, most EOL processing options necessitate a certain level of disassembly. *Disassembly* is the process of the 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 *selective* (product not fully disassembled) or *complete* (product fully disassembled), and may use a methodology that is *destructive* (focusing on materials rather than components recovery) or *non-destructive* (focusing on components rather than materials recovery). Disassembly operations are very complex, time-consuming and expensive. Recent books by Lambert and Gupta [13] and McGovern and Gupta [17] can be helpful in understanding the general area of disassembly.

As mentioned before, EOL processing often necessitates a certain level of disassembly (an expensive process due to its labor-intensive nature). Hence, finding an efficient disassembly sequence is necessary. In addition, the complexity of determining the best disassembly sequence increases as the number of parts of a product grows. In fact, this problem has been proven to be NP complete [16]. Thus, not only an efficient methodology is required for disassembly sequencing, but also limiting the disassembly operations to recyclable materials and reusable components in the EOL product is crucial to making the recovery operations economically viable.

In recent years, genetic algorithm (GA) has been gaining popularity for solving combinatorial and NP-complete problems [11]. 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.

A GA for multi-objective optimization was proposed by Valenzuela-Rendón and Uresti-Charre [23] who calculated the fitness of each entity in the population incrementally based on the degree to which it was dominated or how close it was to other entities. The behavior of each was 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 commented that GA had a stable and reliable time response. Keung et al. [10] applied a GA approach to a tool selection problem. In their paper, the overall objective of the model was to minimize the processing time. Loughlin and Ranjithan [15] proposed a GA method, to a so-called neighborhood constraint problem, and concluded that the GA performed better in multi-objective problems compared to single objective problems. Lazzerini and Marcelloni [14] used GA in scheduling assembly processes. They employed modified partially matched crossover (PMX) and mutation operations to obtain a near optimal sequence. The precedence relationships were not considered in their model.

One of the factors that add to complications in sequencing problems is precedence relationships. The conventional search algorithms often use combinatorial search techniques and then augment them with precedence relationships. Sanderson et al. [21] considered precedence relationships in assembly sequence planning in such a manner. Regular GAs are generally not suitable for the systems where precedence relationships and constraints are involved. Seo et al. [22], for example, proposed a genetic algorithm for generating optimal disassembly sequences considering both economical and environmental factors. However, their search could lead to infeasible strings during the crossover and mutation operations. The authors addressed that situation by penalizing the string with the hope that it would be eliminated during latter generations. However, that turned out to be a weakness in their algorithm. Bierwirth et al. [3] and Bierwirth and Mattfeld [2] proposed a methodology to overcome such a 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. While Bierwirth et al. [3] and Bierwirth and Mattfeld [2] employed the PPX approach to a single objective job shop scheduling problem, it works equally well for multiple criteria modeling [11], where the authors tackled the complete disassembly problem. However, as was mentioned above, it is imperative to develop an efficient methodology by limiting disassembling operations only to recyclable materials and reusable components in the EOL product so that the recovery operations are economically viable. To that end, in this paper, we present a GA-based methodology to perform selective disassembly of a product. This is a generalized methodology that not only can determine sequences for selective disassembly, but it can also do the same for complete disassembly.

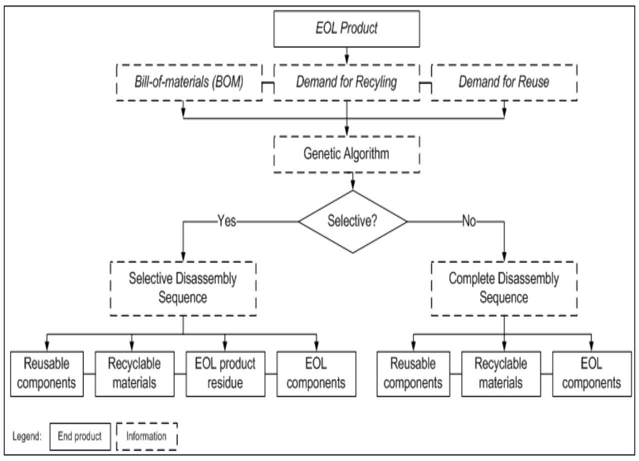


Figure 1: Disassembly sequencing system definition with successive EOL flows.

2 Materials and methods

Proposed disassembly sequencing system requires a bill of materials (BOM), and separate demands for recyclable materials and reusable components in the EOL product (see Figure 1). BOM data include fastener type, disassembly time, and coordinates, and the precedence relationships in the product structure. After the data is embedded in the genetic algorithm (GA), the user is prompted for the type of disassembly operation (viz. selective or complete disassembly). In the case where complete disassembly is selected, the EOL product is disassembled completely, leaving recyclable materials, reusable components and EOL components behind. If selected disassembly is the required method, only the items demanded for reuse and recycling are taken out of the product structure. Since some of the components must be disassembled due to precedence relationships regardless of their corresponding demands, selective disassembly leaves behind an EOL product residue in addition to the EOL components.

2.1 Nomenclature

The notations used in the rest of the paper are given in the table below.

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 degree change is required, 2: if 180 degree 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 ($rnd = 1, \ldots, 9$)
seq	Index for disassembly sequence ($seq = 1, \ldots, 9$)
T_{seq}	Cumulative disassembly time after component j in sequence *seq* is disassembled (s)

2.2 Disassembly sequence generation for EOL selective disassembly operations

Several metaheuristic applications have been utilized to solve disassembly sequencing problems, such as expert systems, simulated annealing, Petri nets and neural networks [9,12]. In particular, genetic algorithm (GA) is often used for its capability to evolve towards optimal solution without processing all the alternatives [6,7].

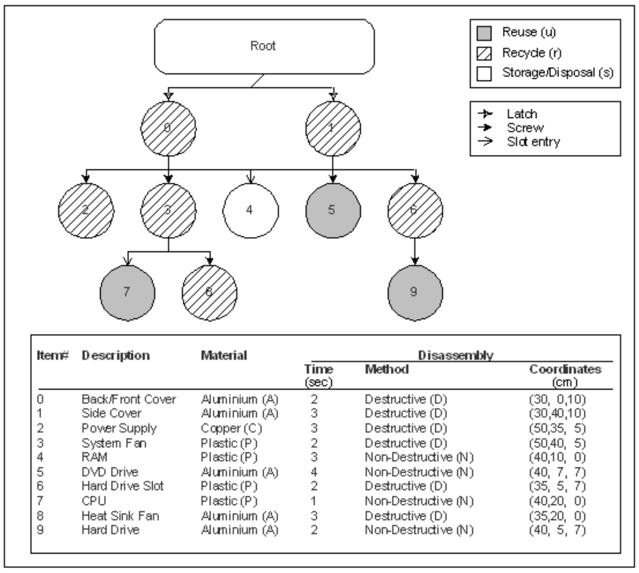
This paper utilizes a version of genetic algorithm (GA) to generate feasible sequences for selective disassembly. GA starts with a set of randomly selected potential solutions called the *population*. Each member of the population is encoded as an artificial *chromosome* which contains information about the solution mapping. A custom *fitness function* is designed so that the fitness score of each chromosome is individually calculated. The chromosomes are evaluated according to their fitness scores and are iteratively regenerated to increase their fitness in the next generation. In every generation, there is a probability that a *mutation* will occur in one or more chromosomes. In addition, *crossover*, the exchange of genetic information between two mating chromosomes, may also occur. The selected fittest chromosomes are further processed. If any of the predetermined termination conditions is met, the genetic algorithm terminates [4].

Basic structure of a genetic algorithm is as follows [adopted from [18]]:

```
gn := 0;
Compute initial feasible random population;
WHILE stopping condition not fulfilled DO
BEGIN
        Select individuals for reproduction;
        Conduct crossover operations to generate new
        individuals;
        Conduct mutation operations to mutate some
        individuals;
        Compute new generation
END
```

2.3 Elements of genetic algorithm for selective disassembly sequencing

The bill of materials (BOM) of the EOL product and related data are provided in Figure 2. The given product consists of ten components ($n = 10$) indexed by integers from 0 to 9 (see Figure 3). Therefore, $j \in \{0, 1, \ldots, n-1\}$. The location of each component is defined by its corresponding 3D coordinates. The methodology 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



Item#	Description	Material	Disassembly		
			Time (sec)	Method	Coordinates (cm)
0	Back/Front Cover	Aluminium (A)	2	Destructive (D)	(30, 0,10)
1	Side Cover	Aluminium (A)	3	Destructive (D)	(30,40,10)
2	Power Supply	Copper (C)	3	Destructive (D)	(50,35, 5)
3	System Fan	Plastic (P)	2	Destructive (D)	(50,40, 5)
4	RAM	Plastic (P)	3	Non-Destructive (N)	(40,10, 0)
5	DVD Drive	Aluminium (A)	4	Non-Destructive (N)	(40, 7, 7)
6	Hard Drive Slot	Plastic (P)	2	Destructive (D)	(35, 5, 7)
7	CPU	Plastic (P)	1	Non-Destructive (N)	(40,20, 0)
8	Heat Sink Fan	Aluminium (A)	3	Destructive (D)	(35,20, 0)
9	Hard Drive	Aluminium (A)	2	Non-Destructive (N)	(40, 5, 7)

Figure 2: EOL product structure, BOM and other data.

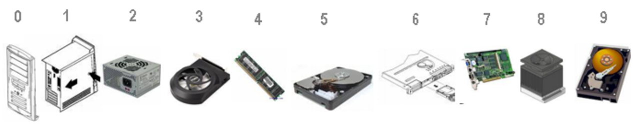


Figure 3: Components of the product.

Sequence	Method	Demand	Material	$F(ch, gn)$
0623795418	DDDDNNNNDD	rrrruuusrr	APCPPAAPAA	28.978
0613275948	DDDDDNNNND	rrrrruuusr	APAPCPAAPA	29.881
1630824957	DDDDDDNNNN	rrrrrrsuuu	APPAACPAAP	30.563
0327468195	DDDNNDDDNN	rrrusrrruu	APCPPPAAAA	30.586
1328047659	DDDDDNNDNN	rrrrrsuruu	APCAAPPPAA	30.813
		...		
1385206479	DDDNDDDNNN	rrrurrrsuu	APAACAPPPA	35.052
0463159827	DNDDDNNDDN	rsrrruurru	APPPAAAACP	35.137
1342760598	DDNDNDDNND	rrsrurruur	APPCPPAAAA	35.142
1385476209	DDDNNNDDDN	rrrusurrru	APAAPPPCAA	35.296
0213657984	DDDDDNNNDN	rrrrruuurs	ACAPPAPAAP	35.430

Table 1: Initial population.

demanded, it is represented by s. If it is demanded, it may be demanded for reuse (u) or recycling (r). A component has one of three types of joints, namely, latch (L), screw (S) or slot entry (E). In addition, the precedence relationships are given as follows: components 1 **and** 2 must be disassembled prior to any other component; component 3 must be disassembled prior to components 7 **and** 8; and component 6 must be disassembled prior to component 9.

Chromosomes
GA requires that the solution and parameters be coded into chromosomes, represented by a combination of numbers, alphabets and/or other characters, before they can be processed. In this study, in order to capture the five variables, chromosomes are codified in the form of a string consisting of five ordered sections of equal length, representing the disassembly sequence, the disassembly method, the demand type of each component, and the material type of each component, respectively. Coordinate data are kept separate from the chromosome structure to provide ease in calculations. For instance, in the sample chromosome provided below:

062**3**795418 DDD**D**NNNNDD rrr**r**uuusrr APC**P**PAAPAA

item 3 requires destructive disassembly (D), is demanded for recycling (r) and is made out of plastic (P).

Initial population
The initial population consists of ncr random chromosomes. The population preserves the precedence relationships and other constraints imposed by the product structure. For the example provided in Figure 2, hundred chromosomes (feasible solutions) are randomly created to form the initial population ($ncr = 100$). The chromosomes in the initial random population are provided in Table 1 (only 10 out of 100 are shown).

Crossover
The proposed algorithm employs the precedence preservative crossover (PPX) methodology for crossover. In this methodology, in addition to the two strings representing the chromosomes of the parents (Parent$_1$ and Parent$_2$), two additional strings pass on the precedence relationship based on the two parental permutations to two new offsprings while making sure that no new precedence relationships are introduced. A vector, representing the number of operations involved in the problem, 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.

For instance, consider two chromosome strings (Parent$_1$ and Parent$_2$) provided below:

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

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

Assuming that the two random masks created from the above parents are

Mask$_1$: II II II I I II II II I II,

Mask$_2$: II I I II I II II I II II,

the output of the crossover process (viz. Child$_1$ and Child$_2$) is generated as given below:

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

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

Mutation
The mutation occurs with a pre-determined probability. If the probability holds, the mutation operator selects a random number of genes ($rnd = 1, \ldots, 9$), 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 0 and 1. The rest of the strings remain the same. For instance, assuming that $rnd = 3$, the first three chromosomes in the population mutate (see Table 2).

Fitness evaluation
The fitness function is dependent on the total disassembly time. There are three factors that contribute to the disassembly time. 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 follows:

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

Chromosome number	Before mutation	After mutation
1	0623795418	1623795408
2	0613275948	1603275948
3	1630824957	0631824957
4	0327468195	0327468195
5	1328047659	1328047659
6	1385206479	1385206479
7	0463159827	0463159827
8	1342760598	1342760598
9	1385476209	1385476209
10	0213657984	0213657984

Table 2: An example of the proposed mutation operation.

The second factor ($ct_{j,seq}$) is the penalty (in seconds) for each travel time to disassemble component j in sequence *seq*, which is a function of the distance traveled between the $(seq-1)^{\text{th}}$ and seq^{th} sequences and the robot arm speed factor (sf):

$$ct_{j,seq} = \frac{\sqrt{(x_{j,(seq-1)}-x_{j,seq})^2+(y_{j,(seq-1)}-y_{j,seq})^2+(z_{j,(seq-1)}-z_{j,seq})^2}}{sf}.$$

The final factor in fitness function is the penalty for disassembly method change ($mt_{j,seq}$). For each disassembly method change, the sequence is penalized by 1 second:

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

The algorithm searches for a “recycling pair” and does not penalize the sequence if the two adjacent components are made of the same material and if they are both demanded for recycling.

Let T_{seq} denote the cumulative disassembly time after the disassembly operation in sequence *seq* is completed for component j:

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

The objective of the GA model is to minimize the total fitness function (F) by minimizing (i) the traveled distance, (ii) the number of disassembly method changes and (iii) by combining the identical-material components together, eliminating unnecessary disassembly operations. Let $F(ch, gn)$ denote the total fitness for chromosome *ch* in generation *gn*. Hence, total time to disassemble all the components can be calculated as follows:

$$\begin{aligned} F(ch, gn) &= \sum_{seq=0}^{n-1} dt_{j,seq} + \sum_{seq=0}^{n-2} ct_{j,seq} \\ &\quad + \sum_{seq=0}^{n-2} mt_{j,seq}, \quad \forall j,\ j = 0, \dots, n-1. \end{aligned}$$

Selection and regeneration procedure

Following each generation, the chromosomes obtain a certain expectation depending on their fitness values. A roulette wheel is then implemented to select the sequence of parents that will be included in the next generation (the higher the fitness value the higher the chance to be selected). This method aims at allowing the parents in the current generation to be selected for the next generation without getting trapped in the local optima. In addition, a new population is generated eliminating the weak chromosomes.

Termination

The execution of GA terminates if the number of generations reaches up to a maximum value (100 in our example).

Remark

The proposed algorithm allows the user to enter his/her decision regarding selective (partial) or complete disassembly. In selective disassembly, only the parts that are indicated by the user are disassembled via the GA algorithm while complete disassembly would take apart all components in the bill of materials. Regardless of the selection, the precedence relationships are preserved throughout the disassembly sequence generation. In selective disassembly, there is no penalty for the components that are not taken out since the proposed fitness function takes the overall disassembly time into account and does not consider cost and revenue measures such as holding cost, and/or storage cost, and so forth. In addition, regardless of the user selection, the algorithm requires at least three components to be disassembled for the sequence generation to be valid.

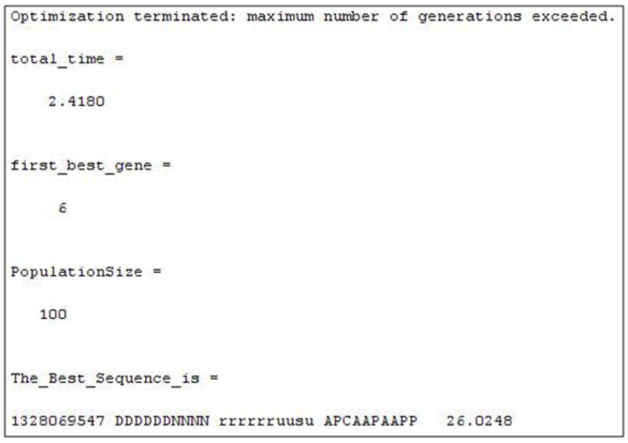
3 Results and discussion

The crossover and mutation probabilities are assumed to be 0.60 and 0.005 respectively for the product structure provided in Figure 1. Initial population consists of 100 chromosomes ($ncr = 100$). After the GA is run for the case of complete disassembly, only one optimal solution is obtained in the final population with a fitness function value of 26.0248 seconds (see Figure 4).

It took 73.8821 seconds to determine the optimal solution using exhaustive search whereas it took only 2.4180 seconds when genetic algorithm was used (see Figure 4). The genetic algorithm reached the solution at the 6th generation. The initial solution of 100 chromosomes included four identical chromosomes, whereas the rest of them were unique sequences. It is important to note that the optimal sequence is found in a few iterations even though it was not present in the initial random population.

The proposed model assumes that the end effector speed for the robot arm is a constant value of 25 cm/s. In addition, the time spent for robot arm angle change (for all three angles) is assumed to be embedded in the disassembly

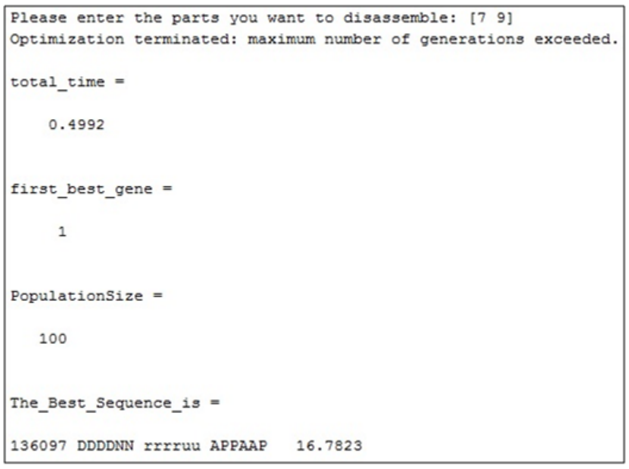
Sequence	***F(ch,gn)***
1328069547 DDDDDDNNNN rrrrrruusu APCAAPAAPP	26.0248 secs



```
Optimization terminated: maximum number of generations exceeded.

total_time =

    2.4180


first_best_gene =

     6


PopulationSize =

   100


The_Best_Sequence_is =

1328069547 DDDDDDNNNN rrrrrruusu APCAAPAAPP   26.0248
```

Figure 4: Matlab screenshot of the complete disassembly sequencing result.

Sequence	***F(ch,gn)***
136097 DDDDNN rrrruu APPAAP	16.7823 secs



```
Please enter the parts you want to disassemble: [7 9]
Optimization terminated: maximum number of generations exceeded.

total_time =

    0.4992


first_best_gene =

     1


PopulationSize =

   100


The_Best_Sequence_is =

136097 DDDDNN rrrruu APPAAP   16.7823
```

Figure 5: Matlab screenshot disassembly sequencing result for the selective disassembly of components 7 and 9.

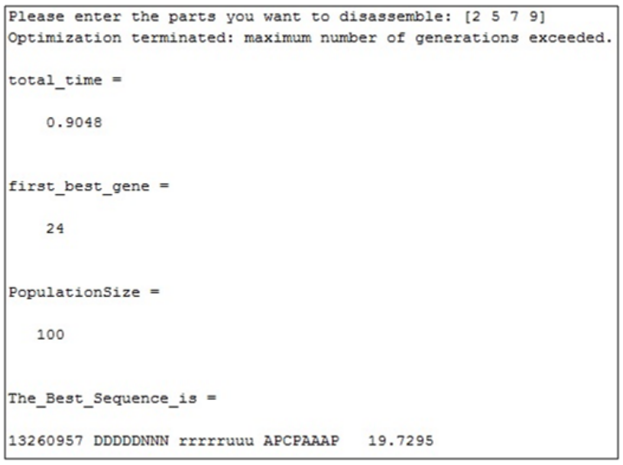
time for each component. In addition, every component is assumed to have one joint that connects the component to the rest of the product structure.

The algorithm is run hundred times to obtain average computation time. On an average, the optimal solution was reached in 2.5576 seconds at the 16.56th generation with an average fitness value of 26.4013 seconds.

The algorithm is coded in Matlab (version $7.7.0.471$ (R2008b)) using Genetic Algorithm and Direct Search Toolbox. The code was run on a computer with a Processor Intel Core 2 Duo CPU P8400 2.26 GHz, 4.00 GB RAM.

For the selective disassembly, components 7 **and** 9 and components 2, 5, 7 **and** 9 are selected for two partial

Sequence	***F(ch,gn)***
13260957 DDDDDNNN rrrrruuu APCPAAAP	19.7295 secs



```
Please enter the parts you want to disassemble: [2 5 7 9]
Optimization terminated: maximum number of generations exceeded.

total_time =

    0.9048


first_best_gene =

    24


PopulationSize =

   100


The_Best_Sequence_is =

13260957 DDDDDNNN rrrrruuu APCPAAAP   19.7295
```

Figure 6: Matlab screenshot disassembly sequencing result for the selective disassembly of components 2, 5, 7 and 9.

disassembly sequences. Figures 5 and 6 depict the results of these sequences, respectively. For the parts 7 **and** 9, the best fitness value (16.7823 seconds) is obtained in the 1st generation in 0.4992 seconds.

For the parts 2, 5, 7 **and** 9, the best fitness value (19.7295 seconds) is obtained in the 24th generation in 0.9048 seconds.

The algorithm is then run 100 times for each of the partial disassembly sequences to obtain the average performance values. The model for generating a disassembly sequence for only the parts 7 **and** 9, on an average, took 0.5370 seconds and obtained the optimal solution at the 1.34th generation. The model for generating a disassembly sequence for only the parts 2, 5, 7 **and** 9, on an average, took 0.6117 seconds and obtained the optimal solution at the 10.83th generation.

These partial disassembly sequencing models did not include penalty for not disassembling the non-demanded parts since the fitness value is a function of the overall disassembly time and does not include any cost/revenue measures such as holding cost and/or disposal cost.

Several proofs have been developed that describe the expected convergence time and provide worst-case and average-case convergence time [1,19,20]. Ankenbrandt [1] demonstrated that, with proportional selection, GAs have average and worst case time complexity in $\mathrm{O}(\mathrm{O}(\mathrm{Evaluate}(X)) * m\frac{\log(m)}{\log(r)})$, where m is the population size, r is the fitness ratio (the average fitness of a chromosome over the average fitness function of all other chromosomes in the generation), and “Evaluate” represents the combined domain-dependent evaluation and decoding functions of the chromosome X (complexity of the fitness, mutation and crossover functions).

The $\mathrm{O}(\mathrm{Evaluate}(X))$ is the order of the combined functions computed in every step. For the fitness function, it is in the order of $\mathrm{O}(n * m)$, where n is the chromosome length. The crossover function is also in the order of $\mathrm{O}(cp * n * m)$, where cp is the crossover probability. Finally, for the mutation function, since it involves two objects from the selected chromosome (regardless of the length of the chromosome), its complexity only depends on the mutation probability which controls the chromosomes selected for mutation. Therefore, the mutation complexity is in the order of $\mathrm{O}(mp * m)$, where mp is the mutation probability.

When comparing them, it is important to note that, even though the complexity functions of both selective and complete disassembly sequencing algorithms are identical, due to the variation of the length of the chromosomes in each algorithm the computational time will vary.

4 Conclusions

A genetic algorithm model is utilized to obtain economically and environmentally sustainable disassembly sequences. The algorithm utilizes BOM data including material, EOL processing option (reuse, recycle, etc.), fastener type, disassembly time, and coordinates, and the precedence relationships in the product structure. The model provides fast and accurate input for the disassembly scheduling environments for both complete and selective disassembly. The GA does 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. Future work will include combining the proposed disassembly algorithm with sensory-driven automated robotic disassembly applications.

References

[1] C. Ankenbrandt, *An extension to the theory of convergence and a proof of the time complexity of genetic algorithms*, in Foundations of Genetic Algorithms, Morgan Kaufman, 1991, 53–68.

[2] C. Bierwirth and D. C. Mattfeld, *Production scheduling and rescheduling with genetic algorithms*, Evolutionary Computation, 7 (1999), 1–18.

[3] C. Bierwirth, D. C. Mattfeld, and H. Kopfer, *On permutation representations for scheduling problems*, in Parallel Problem Solving from Nature–PPSN IV, H. M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, eds., vol. 1141 of Lecture Notes in Computer Science, Springer-Verlag, Berlin, 1996, 310–318.

[4] A. El-Sayed, E. Kongar, and S. M. Gupta, *A genetic algorithm approach to end-of-life disassembly sequencing for robotic disassembly*, in Proceedings of the 2010 Northeast Decision Sciences Institute Conference, Alexandria, VA, 2010, 402–408.

[5] A. Gungor and S. M. Gupta, *Issues in environmentally conscious manufacturing and product recovery: A survey*, Computers and Industrial Engineering, 36 (1999), 811–853.

[6] S. M. Gupta and P. Imtanavanich, *Evolutionary computational approach for disassembly sequencing in a multiproduct environment*, Journal Biomedical Soft Computing and Human Sciences, 15 (2010), 73–78.

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

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

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

[10] K. W. Keung, W. H. Ip, and T. C. Lee, *The solution of a multi-objective tool selection model using the GA approach*, International Journal of Advanced Manufacturing Technology, 18 (2001), 771–777.

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

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

[13] A. J. D. Lambert and S. M. Gupta, *Disassembly Modeling for Assembly, Maintenance, Reuse, and Recycling*, CRC Press, Boca Raton, Florida, 2005.

[14] B. Lazzerini and F. Marcelloni, *A genetic algorithm for generating optimal assembly plans*, Artificial Intelligence in Engineering, 14 (2000), 319–329.

[15] D. H. Loughlin and S. Ranjithan, *The neighborhood constraint-method: A genetic algorithm-based multiobjective optimization technique*, in Proceedings of the Seventh International Conference on Genetic Algorithms, 1997, 666–673.

[16] S. M. McGovern and S. M. Gupta, *Combinatorial optimization analysis of the unary NP-complete disassembly line balancing problem*, International Journal of Production Research, 45 (2007), 4485–4511.

[17] S. M. McGovern and S. M. Gupta, *The Disassembly Line: Balancing and Modeling*, McGraw Hill, New York, 2011.

[18] D. P. Mukhopadhyay, M. O. Balitanas, A. Farkhod, S.-H. Jeon, and D. Bhattacharyya, *Genetic algorithm: A tutorial review*, International Journal of of Grid and Distributed Computing, 2 (2009), 25–32.

[19] B. Rylander and J. Foster, *Computational complexity and genetic algorithms*, in Proceedings of the World Science and Engineering Society's Conference on Soft Computing, Advances in Fuzzy Systems and Evolutionary Computation, World Science and Engineering Society Press, 2001, 248–253.

[20] B. Rylander, T. Soule, and J. Foster, *Computational complexity, genetic programming, and implications*, in Proceedings of the European Genetic Programming Conference, 2001.

[21] A. C. Sanderson, L. S. Homem de Mello, and H. Zhang, *Assembly sequence planning*, AI Magazine, 11 (1990), 62–81.

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

[23] M. Valenzuela-Rendón and E. Uresti-Charre, *A non-generational genetic algorithm for multiobjective optimization*, in Proceedings of the Seventh International Conference on Genetic Algorithms, 1997, 658–665.