A Heuristic Solution for the Disassembly Line Balancing Problem
Incorporating Sequence Dependent Costs

A.J.D. Lambert\textsuperscript{1} and Surendra M. Gupta\textsuperscript{2}

\textsuperscript{1} Technische Universiteit Eindhoven, P.O. Box 513, 5600 MB Eindhoven NL, TM/AW,
\texttt{a.j.d.lambert@tm.tue.nl}

\textsuperscript{2} Laboratory for Responsible Manufacturing, 334 SN, Department of MIE, Northeastern University,
360 Huntington Avenue, Boston, MA, 02115 USA, \texttt{gupta@neu.edu}

ABSTRACT

This paper deals with disassembly sequencing problems subjected to sequence dependent disassembly costs. We present a heuristic and an iterative method based on partial branch and bound concept to solve such problems. Since heuristic methods intrinsically generate suboptimum solutions, we compared the heuristically obtained solutions with the exact solutions to see if they are reasonably good or not. This process, however, is limited to small or perhaps medium sized problems only as the required CPU time for exact methods tends to increase exponentially with the problem size. For the problems tested, we observed that the methods described in this paper generate surprisingly good results using almost negligible amount of CPU time.

1. INTRODUCTION

End-of life processing of complex products such as PCs is becoming increasingly important, because they contain a large variety of hazardous, useful as well as valuable components and materials. Disassembly is often used to separate such components and materials. A disassembly precedence graph (DPG) is frequently used to describe a disassembly process. Nodes in this graph refer to operations, typically the detachments of components. Arcs represent the precedence relationships. Both yield and costs are associated with every operation. The yield results from the component that is available. The costs are those of labor and capital goods that are required for the respective operation. The costs and yield translate into profit (either positive or negative) which is assigned to every node.

The search space of a modest disassembly precedence graph, which includes all the possible disassembly sequences, is large. It combinatorially explodes with increased product complexity, which is typically related to its number of components. Fortunately, one can rely on linear programming methods if the cost of a single operation is sequence independent. In such case, extended problems can be solved at the expense of CPU time. In practice, however, costs are typically sequence dependent, because any operation might include sequence dependent tasks such as tool selection, product and tool reorientation, etc. In such case, linear programming no longer offers an adequate tool for the selection of the optimum disassembly sequence. The solution of this problem intrinsically requires integer linear programming methods, which are far less efficient in CPU time. This is one of the reasons why many researchers rely on heuristic and metaheuristic methods. Heuristic methods, in particular, can be efficient tools for achieving the so-called "good enough" solutions. However, here a serious shortcoming is that there is no tool available to determine the goodness of such solution. In other words, it is often unclear whether a "good enough" solution is really good enough. This makes the exact method an indispensable tool to develop, improve and adjust heuristic methods as well as to benchmark the solution obtained using heuristic methods.

2. LITERATURE REVIEW

The sequencing problem with sequence dependent costs has been discussed in the literature. The first application of the rigorous exact method to a DPG was discussed by Johnson and Wang (1998). The authors used a two-commodity network flow approach, which is based on integer linear programming and is related to a constrained Traveling Salesman Problem. This method was extended to an AND/OR graph by Kang \textit{et al.} (2001). An adapted method with a reduced set of integer variables has been discussed by Lambert (2005c). Binary integer linear programming (BILP) has
been applied by Yee and Ventura (1999) to AND/OR graphs without parallelism and, consequently, without the possibility of short tours. The authors also proposed a metaheuristic (Lagrangian relaxation) for extended problems. Iterative BILP has been adapted by Lambert (2005a) for solving general problems with AND/OR graphs, and by Lambert (2005b) for DPGs. Metaheuristic methods are compiled by Santochi et al., (2001). Gungor and Gupta (1997, 2001) and Rosell et al. (2003) have presented heuristic methods. Combination of exact and heuristic methods has been proposed by Lambert (2004). For a thorough coverage on disassembly modeling, see Lambert and Gupta (2005b).

3. THE EXAMPLE PRODUCT

The methods that are presented in this paper for solving the sequencing problem will be demonstrated by implementing them to a case example. For this, we selected a cell phone, which has already been extensively studied for testing a heuristic disassembly line-balancing algorithm. This 25 components product has been adapted from Gupta et al. (2004) and is depicted in Figure 1.

![Figure 1. Interior components, front cover, and keyboard of the cell phone.](image)

Each of the 25 components in the product refers to the disassembly operation of its disestablishment. The 25 physical operations are supplemented by an extra nonphysical operation, which is called the combined source/sink operation and is represented by the index zero. The operations are as follows:

0. Source (start process)/sink (finish process)
1. Removing antenna
2. Removing battery
3. Removing bolt #1 type 1
4. Removing bolt #2 type 1
5. Discarding antenna path
6. Removing bolt #1 type 2
7. Removing bolt #2 type 2
8. Removing bolt #3 type 2
9. Removing clip
10. Removing bolt #4 type 2
11. Disconnecting red/blue cable
12. Removing metal top
13. Disconnecting orange cable
14. Removing rubber
15. Disconnecting white cable
16. Removing black back
17. Removing speaker
18. Removing front cover
19. Removing circuit board
20. Removing plastic screen
21. Removing keyboard  
22. Removing microphone  
23. Discarding LCD  
24. Removing board under keyboard  
25. Removing inside black board

Directed arrows in Figure 2 depict the precedence relationships between these operations. For example, operation 15 can only be performed after operations 6, 7, 8, and 10 have been performed.

In order to study the sequence dependent disassembly, we introduce the concept of profit per operation (the difference between yield and cost of every operation). Because the costs are sequence dependent, the profits are thus also sequence dependent. By randomly generating profit values, $\Pi_{j,k}$, varying from 1 to 99 per operation and arranging them in a matrix $\Pi$, we can arrive at an unbiased instance of the problem. Here, $\Pi_{j,k}$ corresponds to the profit that is obtained when operation $k$ is performed followed by operation $j$. A typical instance of a profit matrix is presented in Figure 3. A value of zero in a matrix cell refers to an infeasible subsequence.

![Disassembly Precedence Graph for the cell phone.](image)

Figure 3. An instance of the profit matrix, $\Pi$, for the cell phone.
4. THE HEURISTIC METHOD

4.1. Extended greediness

Heuristic methods are based on a set of prescribed rules. Often, these rules are derived from practical experiences and intuitions. For a network representation of the disassembly process, one can follow a more formal approach. For a typical disassembly precedence graph we discuss a forward approach, starting with the nonphysical source node 0 (where the disassembly process starts).

The disassembly sequence ends with the nonphysical sink node, which is also given a node number of 0. A value of zero is assigned to cells \( \Pi_{j,0} \) for all \( j \). Note that certain cells in the \( \Pi \) matrix are never visited because they are inhibited by the set of precedence relationships. This, for example, holds true for all the diagonal elements \( \Pi_{k,k} \). For the DPG of Figure 2, it is similarly clear, for example, that \( \Pi_{1,10} \) is not feasible, as it is apparent that operation 10 cannot be performed immediately after operation 1.

We could uncover an enumerative procedure to find the most profitable disassembly sequence by starting with the expansion of the source node. It is clear from the DPG (Figure 2) that the source can be expanded in five subsequences, viz., 0-0, 0-1, 0-2, 0-3, and 0-4. Using the instance of Figure 3, the respective profits of 0, 40, 84, 25, and 69 are achieved. Note that the subsequence that ends with the sink node, viz., 0-0, completes the sequence. This corresponds to the case of partial disassembly (here, however, it means that no disassembly is carried out at all). This sequence results in a zero profit.

The remaining four subsequences are further expanded, resulting in 19 additional subsequences, viz., 0-1-0, 0-1-2, 0-1-3, 0-1-4, 0-2-0, 0-2-1, 0-2-3, 0-2-4, 0-2-6, 0-2-7, 0-2-8, 0-3-0, 0-3-1, 0-3-2, 0-3-4, 0-4-0, 0-4-1, 0-4-2, 0-4-3. Proceeding by further expanding the remaining subsequences, we would have supposedly enumerated all possible disassembly sequences. However, the set of all the possible disassembly sequences include a number of sequences that are virtually inaccessible to enumerative calculations.

We will demonstrate this phenomenon by considering the product represented by the DPG of Figure 2 and splitting it into two subgraphs, viz., the smaller one with operations 3, 4, 9, 11, and 16 and the larger one with the rest of the operations. Consider the following feasible sequence (one of many feasible sequences) of operations belonging to the larger subgraph:

\[
0-1-2-7-5-6-8-10-15-14-18-13-12-17-21-20-19-23-22-24-25-0
\]

Note that the only possible sequences for the smaller subgraph as follows:

3-4-9-11-16 or 4-3-9-11-16.

These operations of the smaller subgraph can be placed at arbitrary positions in the sequence of the larger subgraph to obtain a feasible sequence of the DPG of Figure 2, provided the order of sequence of the smaller subgraph is maintained. The following are three (of many) such examples:

\[
0-3-\textbf{4-9-11-16}-1-2-7-5-6-8-10-15-14-18-13-12-17-21-20-19-23-22-24-25-0
\]

or:

\[
0-1-2-7-5-6-8-10-15-14-18-13-12-17-21-20-19-23-22-24-25-\textbf{3-4-9-11-16}-0
\]

or:

\[
0-1-3-2-7-5-6-8-4-10-15-9-14-18-13-12-17-21-\textbf{11}-20-19-23-22-16-24-25-0
\]

While there is no available method for determining the size of the search space of an arbitrary DPG, a method is available that holds true for completely divergent, multi-level DPGs (Uchiyama et al, 1994). It can be shown that for the specific sequence of the larger subgraph considered above, there are 106,260 possible sequences that could be derived using the two feasible sequences of the smaller subgraph given above (see also Lambert, 2005b). Bearing in mind that the larger subgraph results in a multitude of possible sequences, and knowing that merging any of these sequences with
the smaller subgraph results in over 100,000 possible sequences, it is clear that the order of magnitude of the search space is indeed substantial.

It is therefore evident that the enumerative calculation is not a realistic option for the size of the problem considered here. This implies that we must limit the amount of expansion. The other extreme of complete enumerative approach is the greedy approach, where, after expansion, only the most profitable subsequence is considered for further expansion. The greedy sequence that results from the instance of Figure 3 of the cell phone is as follows:

0-2-6-8-3-7-4-9-11-16-15-10-12-14-15-18-13-21-17-19-20-23-24-25-22-0

The corresponding profit is 1720, which can be easily verified via manual calculations. However, one wonders whether this is a 'good enough' solution or not. By introducing a greediness parameter, \( \Lambda \), which is the number of subsequences that should be considered for further expansion, we can extend this method. For instance, with \( \Lambda = 2 \), we select 0-2 and 0-4 for further expansion and from the resulting subsequences we proceed with the best and the second best of the resulting subsequences. Obviously, a further increase in the value of \( \Lambda \) is possible. The results for the case of Figure 3 for increasing values of \( \Lambda \) are compiled in Table 1.

It can be observed that the profit usually increases with increasing values of \( \Lambda \). It must, however, be noted that this increase is non-monotonic. This is because promising subsequences can be selected prematurely, which do not further increase the profit as expansion continues, thus inhibiting the discovery of sequences with better profits.

<table>
<thead>
<tr>
<th>( \Lambda )</th>
<th>Sequence</th>
<th>Profit</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0-2-6-8-3-7-4-9-11-16-15-10-12-14-15-18-13-21-17-19-20-23-24-25-22-0</td>
<td>1720</td>
</tr>
<tr>
<td>2</td>
<td>0-2-6-8-3-7-4-1-9-11-5-10-12-15-18-13-14-17-21-20-23-24-25-19-22-0</td>
<td>1870</td>
</tr>
<tr>
<td>3</td>
<td>0-2-6-8-3-7-4-1-9-11-5-10-12-16-15-18-13-14-17-21-20-23-24-19-22-0</td>
<td>1902</td>
</tr>
<tr>
<td>4</td>
<td>0-2-6-8-3-7-4-1-9-11-5-10-12-16-13-15-17-19-18-14-21-20-23-24-25-22-0</td>
<td>1901</td>
</tr>
<tr>
<td>5-6</td>
<td>0-2-6-8-3-7-4-1-9-11-5-10-12-16-13-14-15-18-17-21-20-23-24-19-25-22-0</td>
<td>1943</td>
</tr>
<tr>
<td>7-8</td>
<td>0-2-6-8-3-7-4-1-9-11-5-10-12-14-15-18-13-21-17-16-20-23-24-19-25-22-0</td>
<td>1965</td>
</tr>
<tr>
<td>9</td>
<td>0-2-6-8-3-7-4-1-9-11-5-10-12-16-13-14-15-18-17-21-20-23-24-19-22-0</td>
<td>1943</td>
</tr>
<tr>
<td>10-271</td>
<td>0-2-6-8-3-7-4-1-9-11-5-10-12-14-15-18-13-21-17-16-20-23-24-19-25-22-0</td>
<td>1965</td>
</tr>
<tr>
<td>( \geq ; 272 )</td>
<td>0-2-1-6-8-5-7-10-3-12-4-14-15-18-13-9-11-17-16-20-23-24-19-21-25-22-0</td>
<td>1977</td>
</tr>
</tbody>
</table>

Indeed, as \( \Lambda \) approaches infinity, the optimum solution will be obtained. This calculation would coincide with the enumerative approach and is obviously not realistic. In our specific case, software restrictions confined us to a maximum value of \( \Lambda \) equaling 277. In this case, the needed CPU time is about 2 sec. From the table above, we conclude that the greedy solution is certainly not 'good enough' as it can be easily surpassed by approximately 16%. Unfortunately, we do not know for sure if the solution that results in the profit of 1977 is 'good enough' or not, as it is basically impossible to increase \( \Lambda \) to the extent that is needed to find that. Fortunately, we had exact methods at our disposal that were able, at the expense of considerable CPU time, to confirm that 1977 was indeed the maximum profit. For more complex problems, however, it is not likely that the exact methods can be applied, because of the associated exponentially increasing CPU time.

4.2. Partial branch-and bound

In an effort to achieve a better solution (that is closer to optimum), the feasibility of partial branch-and bound is considered. This heuristic proceeds as follows. First, we define a parameter \( \Gamma \), which is the number of initial subsequences that are considered at every expansion step. The larger the value of \( \Gamma \), the more accurate the result of the calculation will become. Obviously, \( \Gamma = 1 \) corresponds to the conventional extended greediness calculation. We will apply the method to the product already discussed. Assume that a value of 200 has been selected for the greediness parameter \( \Lambda \). The result of the first iteration step is the sequence listed in Table 1 with the objective value of 1965. We will investigate this further and see if there are better solutions. Let us select \( \Gamma = 5 \). We expand the source, thus arriving at the subsequences: 0-1, 0-2, 0-3, and 0-4. The corresponding profits of the four sequences found are, respectively: 1930, 1965, 1897, and 1930. These all have to be expanded one operation further, and the calculation is repeated, but with the following initial subsequences:
Only those five of the resulting 15 sequences must be selected that have the highest profit. These start with:

0-1-2, 0-2-1, 0-2-6, 0-2-7, and 0-4-1.

The corresponding profits are: 1930, 1977, 1965, 1942, and 1930. The other branches result in lower profits and are no longer considered. The subsequent steps of the calculation proceed analogously. Obviously, this approach does not rule out the possibility of rejecting potentially good solutions; even the optimum solution might be rejected. However, the method appears to be much more efficient than increasing in the value of $\Lambda$ to an appropriately high value. We encountered no sequence with a higher profit than 1977. However, two additional suboptimum sequences with profit 1972 were detected by this method.

Results of the partial branch-and-bound can be graphically represented (see Figure 4, for example). Here the greediness parameter is 270.

![Graphical representation](image_url)

*Figure 4. Graphical representation (partial) of the partial branch-and-bound approach for the problem of Figure 2.*

In this graph, the corresponding positions in the sequence are vertically arranged. The numbers above the nodes denote the operation number while the numbers below the nodes correspond to profits. For instance, if the extended greedy method is applied to the problem with the initial subsequence predefined by 0-2-1-6-3, a sequence with profit 1957 is returned. Note that, for the sake of clarity, the graph is only partly depicted here.

5. CONCLUSIONS AND RECOMMENDATIONS

We presented a novel heuristic algorithm for detecting 'good enough' solutions to the sequencing problems for DPGs with sequence dependent costs. We applied this method to an instance of an existing problem from practice and found optimum and near optimum solutions. Checking with exact methods pointed out that the optimum solution was, in fact, obtained in addition to a set of near optimum solutions. The required CPU time for the heuristic appeared to be negligible, in contrast with the expense for the exact method that has been applied to the same problem.
It is evident that the application of the iterative exact method cannot be extended to arbitrary product complexity. Fortunately, the heuristic method can be successfully applied in such cases as it does not deal with erroneous candidate solutions. The needed CPU time has polynomial complexity instead of exponential, and the effectivity of the heuristic method can be extended by using the partial branch and bound procedure. Because the heuristic method provides us with a list of suboptimum solutions rather than with a single solution, multiple criteria decision making is possible.

We applied these methods to a variety of problems and were able to arrive at consistent results both for the heuristic and the iterative methods.

In future studies, two directions would be interest, viz., (1) the application of both methods to increasingly complex problems, and (2) the extension of the application of the heuristic method to more advanced problems, such as the disassembly line balancing problem.

APPENDIX: THE HEURISTIC SOFTWARE

A program that supports the heuristic algorithms presented in section 4 has been written in VisualBasic 6.0(2). Apart from some model parameters, the model structure and the instance of the profit matrix can be easily introduced and modified as well.

A.1. Extended greedy heuristic.
Software has been written that supports the extended greediness heuristic. Three constants are defined that can be modified by the user:

- MODELSIZE, which corresponds to the number of physical operations. For the example: MODELSIZE = 25,
- LAMBDA, which is the greediness parameter,
- FINLIST, which is the size of the list of the final solutions. It contains not only the best solution so far, but also a list of next-best suboptimum solutions.

Three lists of sequences are present: The Final list (FINLIST), which presents the solutions, the Intermediate list (INTLIST), which contains the sequences to be expanded, and the Provisional list (PROVLIST), which collects the best sequences that are the result of this expansion. Both PROVLIST and FINLIST are sorted.

Sequences are represented by records that consist of two fields: A sequence of operations, and a variable that represents the objective. Expanded sequences include an extra field, viz., a sequence of Boolean variables that are TRUE if the position corresponds to a possible subsequent operation, and FALSE in the opposite case.

The expansion procedure is performed via a subroutine that starts with a specific sequence and that fills the sequence of Booleans. The precedence relationships of the model are included in this subroutine. For this purpose, flow variables are applied. These are Booleans that are TRUE if the corresponding operation is performed, and FALSE in the opposite case.

There is a precedence relationship for every node, e.g., in the DPG of Figure 2 we have:

\[
\text{IF } (\text{flow}(14) = \text{TRUE}) \text{ AND } (\text{flow}(23) = \text{TRUE}) \text{ THEN } (\text{flow}(24) = \text{TRUE})
\]

Obviously, it is easy to modify the model structure.

The profit is updated for every expansion step of any sequence, adding the value of the relevant position of the profit matrix to the already known partial profit. The result is added to PROVLIST. If all the sequences in INTLIST have been expanded one position, the procedure is finished, and the content of PROVLIST is transferred to FINLIST. PROVLIST contains LAMBDA sequences. The update procedure of FINLIST supports incomplete disassembly, because the sequences are sorted according to their profit. If additional operations result in lower profit, the resulting sequences either end up at a lower rank on the list, or they are rejected.
After a number of loops that is determined by MODELSIZE, the program stops and presents a list of the best sequences detected so far. Because not just one but a list of sequences is generated, we are provided with an extended list of suboptimum, but rather good, solutions. The number of these solutions is defined by the constant FINSIZE.

A.2. Partial branch-and-bound

So far, partial branch-and-bound is not completely supported by the software. However, an initialization subroutine is added that modifies the model so far that any desired number of initial operations in a sequence can be predetermined. The profit is automatically adapted to the desired initialization. Therefore, an extra constant, INITCOUNT, is introduced, which equals the number of operations that is predefined. When no partial branch and bound is applied, INITCOUNT = 0. Due to the subroutine, the program will run multiple times for a single value of INITCOUNT. The various subsequences that follow from the preceding step are initialized in such a multiple run. The most promising of the resulting sequences are selected for initializing one step further. Thus, graphs such as the one in Figure 4 are composed.

REFERENCES

Lambert, A.J.D., 2005b, Exact methods in optimum disassembly sequence search for problems subject to sequence dependent costs, Omega (forthcoming).