



ELSEVIER

Available online at www.sciencedirect.com

SCIENCE @ DIRECT®

European Journal of Operational Research xxx (2005) xxx–xxx

EUROPEAN
JOURNAL
OF OPERATIONAL
RESEARCHwww.elsevier.com/locate/ejor

A balancing method and genetic algorithm for disassembly line balancing

Seamus M. McGovern, Surendra M. Gupta *

Laboratory for Responsible Manufacturing, Department of Mechanical and Industrial Engineering, Northeastern University,
334 SN, 360 Huntington Avenue, Boston, MA 02115, USA

Received 8 April 2004; accepted 9 March 2005

Abstract

Disassembly activities take place in various recovery operations including remanufacturing, recycling and disposal. The disassembly line is the best choice for automated disassembly of returned products. It is therefore important that the disassembly line be designed and balanced so that it works as efficiently as possible. The disassembly line balancing problem seeks a sequence which: is feasible, minimizes workstations, and ensures similar idle times, as well as other end-of-life specific concerns. However finding the optimal balance is computationally intensive with exhaustive search quickly becoming prohibitively large even for relatively small products. In this paper the problem is mathematically defined and proven NP-complete. Additionally, a new formula for quantifying the level of balancing is proposed. A first-ever set of a priori instances to be used in the evaluation of any disassembly line balancing solution technique is then developed. Finally, a genetic algorithm is presented for obtaining optimal or near-optimal solutions for disassembly line balancing problems and examples are presented to illustrate implementation of the methodology. © 2005 Elsevier B.V. All rights reserved.

Keywords: Genetic algorithm; Disassembly; Disassembly line balancing; Combinatorial optimization; Product recovery

1. Introduction

In recent years, the implementation of extended manufacturer responsibility together with new, more rigid environmental legislation and increased public awareness has caused a growing number of

manufacturers to begin recycling and remanufacturing their post-consumed products after they are discarded by consumers. In addition, the economic attractiveness of reusing products, subassemblies or parts instead of disposing of them has further fueled this effort. Recycling is defined as a process performed to retrieve the material content of used and non-functioning products. Remanufacturing on the other hand is an industrial process in which worn-out products are restored to like-new

* Corresponding author. Tel.: +1 617 373 4846; fax: +1 617 373 2921.

E-mail address: gupta@neu.edu (S.M. Gupta).

conditions. Thus, remanufacturing provides the quality standards of new products with used parts.

Product recovery aims to minimize the amount of waste sent to landfills by recovering materials and parts from old or outdated products by means of recycling and remanufacturing (including reuse of parts and products). There are many attributes of a product that enhance product recovery; examples include: ease of disassembly, modularity, type and compatibility of materials used, material identification markings and efficient cross-industrial reuse of common parts/materials. The first crucial step of product recovery is disassembly.

Disassembly is a methodical extraction of valuable parts/subassemblies and materials from discarded products through a series of operations. After disassembly, reusable parts/subassemblies are cleaned, refurbished, tested and directed to the part/subassembly inventory for remanufacturing operations. The recyclable materials can be sold to raw-material suppliers while the residuals are sent to landfills.

Disassembly has recently gained a great deal of attention in the literature due to its role in product recovery. A disassembly system faces many unique challenges; for example, it has significant inventory problems because of the disparity between the demands for certain parts or subassemblies and their yield from disassembly. The flow process is also different. As opposed to the normal “convergent” flow in regular assembly environment, in disassembly the flow process is “divergent” (a single product is broken down into many subassemblies and parts). There is also a high degree of uncertainty in the structure and the quality of the returned products. The condition of the products received is usually unknown and the reliability of the components is suspect. In addition, some parts of the product may cause pollution or may be hazardous. These parts tend to have a higher chance of being damaged and hence may require special handling which can also influence the utilization of the disassembly workstations. For example, an automobile slated for disassembly contains a variety of parts that are dangerous to remove and/or present a hazard to the environment such as the battery, airbags, fuel and oil. Various demand sources may also lead to complications in dis-

assembly line balancing. The reusability of parts creates a demand for these parts however, the demands and availability of the reusable parts is significantly less predictable than what is found in the assembly process. Most products contain parts that are installed (and must be removed) in different attitudes, from different areas of the main structure, or in different directions. Since any required directional changes increases the setup time for the disassembly process, it is desirable to minimize the number of directional changes in the final disassembly sequence. Disassembly line balancing is critical in minimizing the use of valuable resources (such as time and money) invested in disassembly and maximizing the level of automation of the disassembly process and the quality of the parts or materials recovered.

In this paper the disassembly line balancing problem (DLBP) is solved using exhaustive search and a combinatorial optimization methodology. Exhaustive search consistently provides the optimal solution, though its exponential time complexity quickly reduces its practicality. Combinatorial optimization techniques however, are instrumental in obtaining optimal or near-optimal solutions to problems with intractably large solution spaces. An exhaustive search algorithm is presented for obtaining the optimal solution to small instances of the DLBP. A genetic algorithm (GA) is then presented for obtaining solutions for the DLBP. The genetic algorithm considered here involves a randomly generated initial population with crossover, mutation and fitness competition performed over many generations. An example from the literature is considered to illustrate the implementation of the methodology. The conclusions drawn from the study include the consistent generation of optimal or near-optimal solutions, the ability to preserve precedence relationships, the superior speed of the algorithm and its practicality due to the ease of implementation in solving disassembly line balancing problems.

2. Literature review

Many authors have discussed the different aspects of product recovery. For a comprehensive

review of environmentally conscious manufacturing and product recovery, see Gungor and Gupta (1999). Brennan et al. (1994) and Gupta and Taleb (1994) have investigated the problems associated with disassembly planning and scheduling. Gutjahr and Nemhauser (1964) first described a solution to the assembly line balancing problem with an algorithm developed to minimize the delay times at each workstation. The heuristic accounts for precedence relationships and seeks to find the shortest path through a network with the resulting technique being similar to dynamic programming. Erel and Gokcen (1964) developed a modified version of Gutjahr and Nemhauser's line balancing problem algorithm by allowing for mixed-model lines (assembly lines used to assemble different models of the same product) by allowing multiple state times, then, during construction of the solution network, considering all state times before categorizing a given set of completed tasks to a workstation. Suresh et al. (1996) first presented a genetic algorithm to provide a near-optimal solution to the assembly line balancing problem that minimized idle time and minimized probability of line stoppage, i.e., minimized the probability that the workstation time exceeds cycle time. Gungor and Gupta (2001, 2002) presented the first introduction to the disassembly line balancing problem and developed an algorithm for solving the DLBP in the presence of failures with the goal of assigning tasks to workstations in a way that probabilistically minimizes the cost of defective parts. McGovern et al. (2003) first proposed applying combinatorial optimization techniques to the DLBP. McGovern and Gupta (2004) later compared various combinatorial optimization techniques for use with DLBP. A recent book by Lambert and Gupta (2005) is helpful in understanding the general area of disassembly.

3. Notation

The following notation are used in the remainder of the paper:

\leq_P polynomial time reduction; polynomial transformation
 $|PRT|$ cardinality of the set of unique part removal times

$|r|$ cardinality of the set of unique part removal directions
 CT cycle time; maximum time available at each workstation
 d_k demand; quantity of part k requested
 D demand rating for a solution; also demand bound for the decision version of DLBP
 D^* lower demand bound (optimal) for a given instance
 D_{nom} upper demand bound (nominal) for a given instance
 F balance for a given solution sequence
 F^* lower balance bound (optimal) for a given instance; also current best balance in population
 F_i balance of chromosome i
 F_{nom} upper balance bound (nominal) for a given instance
 G number of genes (parts for removal)
 h_k binary value; 1 if part k is hazardous, else 0
 H hazard rating for a solution; also hazard bound for the decision version of DLBP
 H^* lower hazard bound (optimal) for a given instance
 H_{nom} upper hazard bound (nominal) for a given instance
 i chromosome identification ($1, \dots, N$)
 I total idle time for a given solution sequence
 I^* lower idle time bound (optimum) for a given instance
 I_i total idle time of chromosome i
 I_{nom} upper idle time bound (nominal) for a given instance
 j workstation count ($0, \dots, NWS_i$)
 k part (gene) identification ($1, \dots, n$)
 n number of parts for removal; also number of genes
 N number of chromosomes (population)
 \mathbb{N} the set of natural numbers ($0, 1, 2, \dots$)
 NHP product's total hazardous part count
 NWS number of workstations required for a given solution sequence
 NWS^* number of workstations lower bound (optimal) for n parts
 NWS_i number of workstations for chromosome i

NWS_{nom}	number of workstations upper bound (nominal) for n parts
p	counter variable
P	the set of n part removal tasks
PRT	part removal time
PRT_{ik}	time required to remove k th part in chromosome i
PRT_k	time required to remove k th part
PS_k	k th part in a solution sequence (i.e., for solution $\langle 3, 1, 2 \rangle$, $PS_2 = 1$)
r	the set of unique part removal directions
r_k	integer value corresponding to the k th part's removal direction
R	direction rating for a given solution sequence; also direction bound for the decision version of DLBP
R^*	lower direction bound (optimal) for a given instance
R_k	binary value; 0 if part k can be removed in the same direction as part $k + 1$, else 1
R_m	mutation rate
R_{nom}	upper direction bound (nominal) for a given instance
R_x	crossover rate
ST_j	station time; total processing time requirement in workstation j
V	maximum range bound for a workstation's idle time
Z	the set of integers $(\dots, -2, -1, 0, 1, 2, \dots)$
Z^+	the set of positive integers $(1, 2, \dots)$
Z_p	p th objective to minimize or maximize

4. Complexity study

In its most basic form the objective of DLBP is to minimize the sum of the idle times at each workstation. Since finding the optimal solution requires investigating all permutations of the sequence of part removal times, there are $n!$ possible solutions. Searching this space is bounded by $O(n^n)$ or exponential time. Since this grows in instance size significantly faster than polynomial time (P), DBLP cannot be optimally solved in P, therefore it is expected that DLBP \notin P.

A language (i.e., problem), L is defined as non-deterministic polynomial time complete (NP-complete) if

1. $L \in NP$, and
2. $\forall L' \in NP, L' \leq_p L$.

Since requirement 1 (L is verifiable in polynomial time, i.e., for DLBP, given a sequence of n parts for removal, verify in $O(n^a)$ time—where a is some constant—that they preserve the precedence constraints and require not more than a given number of workstations) is true, then DLBP $\in NP$. If, in addition, DLBP meets requirement 2, then DLBP is of the class of NP-complete problems and at this time no algorithm is known for which it can be proven to solve the problem in polynomial time. It is easy to check a supposed answer to see if it is correct but a fast way to solve all instances of the problem to optimality is not known.

The decision version of DLBP is NP-complete (and hence, the optimization version is NP-hard). Using the concise style of Garey and Johnson (1979), this is shown in the following proof by restriction to one of the known NP-complete problems (PARTITION). Verification is omitted for brevity; DLBP is easily seen to be solvable in polynomial time by a nondeterministic algorithm. PARTITION is described by:

INSTANCE: A finite set A of tasks, a size $s(a) \in Z^+$ for each $a \in A$.
 QUESTION: Is there a partition $A' \subseteq A$ into two disjoint sets such that $\sum_{a \in A'} s(a) = \sum_{a \in A - A'} s(a)$?

Theorem 1. DLBP is NP-complete.

INSTANCE: A finite set P of tasks, partial order \prec on P , task time $PRT_k \in Z^+$, hazardous part binary value $h_k \in \{0, 1\}$, part demand $d_k \in N$, and part removal direction $r_k \in Z$ for each $k \in P$, workstation capacity $CT \in Z^+$, number $NWS \in Z^+$ of workstations, difference between largest and smallest idle time $V \in N$, hazard measure $H \in N$, demand measure $D \in N$, and direction change measure $R \in N$.

QUESTION: Is there a partition of P into disjoint sets P_A, P_B, \dots, P_{NWS} such that the sum of the sizes of the tasks in each P_X is CT or less, the difference between largest and smallest idle times is

V or less, the sum of the hazardous part binary values multiplied by their sequence position is H or less, the sum of the demanded part values multiplied by their sequence position is D or less, the sum of the number of part removal direction changes is R or less, and it obeys the precedence constraints?

Proof. Clearly DLBP is in NP.

PARTITION \leq_P DLBP. Restrict to PARTITION by allowing only instances in which $CT = \sum_{k=1}^n \frac{PRT_k}{2} \in \mathbf{Z}^+$, $V = 0$, \prec is empty, and $h_x = h_y, d_x = d_y, r_x = r_y, \forall x, y$.

Therefore, DLBP is NP-complete. \square

In complexity theory the class NP-complete is the set of problems that are the hardest problems in NP in the sense that they are the ones most likely not to be in P. If one could find a way to solve an NP-complete problem quickly then that algorithm could be used to solve all NP problems quickly. At present all known optimal algorithms for NP-complete problems (with the exception of certain subproblems using a pseudo-polynomial time algorithm) require time that is exponential in the instance size. It is unknown whether there are any faster algorithms. Therefore, in order to solve an NP-complete problem, one of the following approaches is used (Garey and Johnson, 1979):

- *Approximation:* Application of an algorithm that quickly finds a sub-optimal solution that is within a certain (known) range of the optimal solution,
- *Probabilistic:* Application of an algorithm that provably yields good average runtime behavior for a given distribution of the problem instances,
- *Special cases:* Application of an algorithm which is provably fast if the problem instances belong to a certain special case, and
- *Heuristic:* Application of an algorithm that works reasonably well on many cases, but cannot be proven to always be fast or optimal.

This paper considers an exhaustive search for small instances but proposes a heuristic approach (more accurately, a metaheuristic) for application to all instance sizes.

5. The DLBP model description

The particular application investigated in this paper seeks to fulfill five objectives:

1. Minimize the number of workstations and hence, minimize the total idle time.
2. Ensure workstation idle times are similar.
3. Remove hazardous parts early in the disassembly sequence.
4. Remove high-demand parts before low-demand parts.
5. Minimize the number of part removal direction changes required for disassembly.

A major constraint is the requirement to provide a feasible (i.e., precedence preserving) disassembly sequence for the product being investigated. The result is an integer multi-criteria decision making problem with an exponential search space. DLBP can be modeled as a directed graph (digraph) that is strongly connected when there are no precedence constraints and weakly connected otherwise. Testing a given solution against the precedence relationships fulfills the major constraint of precedence preservation. Minimizing the sum of the workstation idle times, which will also minimize the total number of workstations, attains objective 1 and is described by

$$I = \sum_{j=1}^{NWS} (CT - ST_j), \quad (1)$$

or

$$I = (NWS \cdot CT) - \sum_{k=1}^n PRT_k. \quad (2)$$

This objective is represented as

$$\text{Minimize } Z_1 = \sum_{j=1}^{NWS} (CT - ST_j). \quad (3)$$

Line balancing seeks to achieve perfect balance (all idle times equal to zero). When this is not achievable, either Line Efficiency (LE) or the Smoothness Index (SI) is used as a performance evaluation tool (Elsayed and Boucher, 1994). We

propose a new measure of effectiveness that combines the two and is easier to calculate (McGovern et al., 2003). SI rewards similar idle times at each workstation but at the expense of allowing for a large (sub-optimal) number of workstations. This is because SI compares workstation elapsed times to the largest ST_j instead of CT as this paper's method does. LE rewards the minimum number of workstations but allows unlimited variance in idle times between workstations because no comparison is made between ST_j s. The method proposed in this paper simultaneously minimizes the number of workstations while aggressively ensuring idle times at each workstation are similar. The method is computed based on the minimum number of workstations required as well as the sum of the square of the idle times for all the workstations. This penalizes solutions where, even though the number of workstations may be minimized, one or more have an exorbitant amount of idle time when compared to the other workstations. It provides for leveling the workload between different workstations on the disassembly line. Therefore a resulting minimum performance value is the more desirable solution indicating both a minimum number of workstations and similar idle times across all workstations. This measure of balance is described by

$$F = \sum_{j=1}^{NWS} (CT - ST_j)^2, \quad (4)$$

with the DLBP balancing objective represented as

$$\text{Minimize } Z_2 = \sum_{j=1}^{NWS} (CT - ST_j)^2. \quad (5)$$

Perfect balance is indicated by

$$Z_2 = 0. \quad (6)$$

Note that mathematically, Formula (5) effectively makes Formula (3) redundant due to the fact that it concurrently minimizes the number of workstations. This new method should be effective with traditional assembly line balancing problems as well.

Theorem 2. Let PRT_k be the part removal time for the k th of n parts where CT is the maximum amount of time available to complete all tasks assigned to

each workstation. Then for the most efficient distribution of tasks, the minimum number of workstations, NWS^* satisfies

$$NWS^* \geq \left\lceil \frac{\sum_{k=1}^n PRT_k}{CT} \right\rceil. \quad (7)$$

Proof. If the above inequality is not satisfied, then there must be at least one workstation completing tasks requiring more than CT of time, which is a contradiction. \square

Subsequent bounds are shown to be true in a similar fashion and are presented without proof. The upper bound for the number of workstations is given by

$$NWS_{\text{nom}} = n, \quad (8)$$

therefore

$$\left\lceil \frac{\sum_{k=1}^n PRT_k}{CT} \right\rceil \leq NWS \leq n. \quad (9)$$

The lower bound on F is given by

$$F^* \geq \left(\frac{I}{NWS^*} \right)^2 \cdot NWS^*, \quad (10)$$

while the upper bound is described by

$$F_{\text{nom}} = \sum_{k=1}^n (CT - PRT_k)^2, \quad (11)$$

therefore

$$\left(\frac{I}{NWS^*} \right)^2 \cdot NWS^* \leq F \leq \sum_{k=1}^n (CT - PRT_k)^2. \quad (12)$$

A hazard measure was developed to quantify each solution sequence's performance, with a lower calculated value being more desirable. This measure is based on binary variables that indicate whether a part is considered to contain hazardous material (the binary variable is equal to one if hazardous, else zero) and its position in the sequence. A given solution sequence hazard measure is defined as the sum of hazard binary flags multiplied by their position in the solution sequence,

thereby rewarding the removal of hazardous parts early in the part removal sequence. This measure is represented as

$$H = \sum_{k=1}^n (k \cdot h_{PS_k}) \quad h_{PS_k} = \begin{cases} 1, & \text{hazardous,} \\ 0, & \text{otherwise,} \end{cases} \quad (13)$$

with the DLBP hazardous part objective represented as

$$\text{Minimize } Z_3 = \sum_{k=1}^n (k \cdot h_{PS_k}). \quad (14)$$

The lower bound on the hazardous part measure is given by

$$H^* = \sum_{p=1}^{NHP} p. \quad (15)$$

For example, three hazardous parts would give a best-case value of $1 + 2 + 3 = 6$. The upper bound on the hazardous part measure is given by

$$H_{\text{nom}} = \sum_{p=n-NHP+1}^n p, \quad (16)$$

or

$$H_{\text{nom}} = (n \cdot NHP) - NHP. \quad (17)$$

For example, three hazardous parts in a total of 20 would give an H_{nom} value of $18 + 19 + 20 = 57$ or equivalently, $H_{\text{nom}} = (20 \cdot 3) - 3 = 60 - 3 = 57$. Formula (15)–(17) are combined to give

$$\sum_{p=1}^{NHP} p \leq H \leq \sum_{p=n-NHP+1}^n p = (n \cdot NHP) - NHP. \quad (18)$$

Also, a demand measure was developed to quantify each solution sequence's performance, with a lower calculated value again being more desirable. This measure is based on positive integer values that indicate the quantity required of this part after it is removed—or zero if not desired—and its position in the sequence. Any given solution sequence demand measure is defined as the sum of the demand value multiplied by their posi-

tion in the sequence, rewarding the removal of high demand parts early in the part removal sequence. This measure is represented as

$$D = \sum_{k=1}^n (k \cdot d_{PS_k}) \quad d_{PS_k} \in \mathbb{N}, \forall PS_k, \quad (19)$$

with the DLBP part demand objective represented as

$$\text{Minimize } Z_4 = \sum_{k=1}^n (k \cdot d_{PS_k}). \quad (20)$$

The lower bound on the demand measure (D^*) is given by Formula (19) where

$$d_{PS_1} \geq d_{PS_2} \geq \dots \geq d_{PS_n}. \quad (21)$$

For example, three parts with demands of 4, 5 and 6 respectively would give a best-case value of $(1 \cdot 6) + (2 \cdot 5) + (3 \cdot 4) = 28$. The upper bound on the demand measure (D_{nom}) is given by Formula (19) where

$$d_{PS_1} \leq d_{PS_2} \leq \dots \leq d_{PS_n}. \quad (22)$$

For example, three parts with demands of 4, 5 and 6 respectively would give a worst-case value of $(1 \cdot 4) + (2 \cdot 5) + (3 \cdot 6) = 32$.

Finally, a direction measure was developed to quantify each solution sequence's performance, with a lower calculated value indicating minimal direction changes and a more desirable solution. This measure is based on a count of the direction changes. Integer values represent each possible direction (typically $r = \{+x, -x, +y, -y, +z, -z\}$; in this case $|r| = 6$). These directions are expressed as

$$r_{PS_k} = \begin{cases} +1, & \text{direction} & : +x, \\ -1, & \text{direction} & : -x, \\ +2, & \text{direction} & : +y, \\ -2, & \text{direction} & : -y, \\ +3, & \text{direction} & : +z, \\ -3, & \text{direction} & : -z \end{cases} \quad (23)$$

and are easily expanded to other or different directions in a similar manner. The direction measure is represented as

$$R = \sum_{k=1}^{n-1} R_k \quad R_k = \begin{cases} 1, & r_{PS_k} \neq r_{PS_{k+1}}, \\ 0, & \text{otherwise,} \end{cases} \quad (24)$$

with the DLBP part direction objective represented as

$$\text{Minimize } Z_5 = \sum_{k=1}^{n-1} R_k. \quad (25)$$

The lower bound on the direction measure is given by

$$R^* = |r| - 1. \quad (26)$$

For example, for a given product containing six parts that are installed/removed in directions $r_k = (-y, +x, -y, -y, +x, +x)$ the resulting best-case value would be $2 - 1 = 1$ (e.g., one possible R^* solution containing the optimal, single-change of product direction would be: $\langle -y, -y, -y, +x, +x, +x \rangle$). In the specific case where the number of unique direction changes is one less than the total number of parts n , the upper bound on the direction measure is given by

$$R_{\text{nom}} = |r| \quad \text{where } |r| = n - 1. \quad (27)$$

Otherwise, the measure varies depending on the number of parts having a given removal direction and the total number of removal directions. It is bounded by

$$|r| \leq R_{\text{nom}} \leq n - 1 \quad \text{where } |r| < n - 1. \quad (28)$$

For example, six parts installed/removed in directions $r_k = (+x, +x, +x, -y, +x, +x)$ would give an R_{nom} value of 2 as given by the lower bound of Formula (28) with, for example, a solution sequence of $\langle +x, +x, -y, +x, +x, +x \rangle$. Six parts installed/removed in directions $r_k = (-y, +x, -y, -y, +x, +x)$ would give an R_{nom} value of $6 - 1 = 5$ as given by the upper bound of Formula (28) with, for example, a solution sequence of $\langle -y, +x, -y, +x, -y, +x \rangle$. In the special case where each part has a unique removal direction, the measures for R^* and R_{nom} are equal and given by

$$R^* = R_{\text{nom}} = n - 1 \quad \text{where } |r| = n. \quad (29)$$

Note that the above optimal and nominal balance, hazard, demand, and direction formulae are dependent upon favorable precedence constraints

that will allow for generation of these optimal or nominal measures.

Problem assumptions include:

- Part removal times are deterministic, constant, and integer (or able to be converted to integers).
- Each product undergoes complete disassembly.
- All products contain all parts with no additions, deletions or modifications.
- Each task is assigned to exactly one workstation.
- The sum of the part removal times of all parts assigned to a workstation must not exceed CT.
- Part precedence relationships must be enforced.

6. Generation of benchmark instances

Since DLBP is a recently defined problem, no sets of test instances are known to exist. It was necessary to develop a set of a priori instances for the DLBP to evaluate the efficacy of the exhaustive search algorithm and the genetic algorithm. This was accomplished by using part times consisting exclusively of prime numbers. They were further selected to ensure that no combinations of these part removal times allowed for any equal summations in order to reduce the number of possible optimal solutions. For example, part removal times 1, 3, 5 and 7 and $CT = 16$ would have minimum idle time solutions of not only one 1, one 3, one 5 and one 7 at each workstation, but various additional combinations of these as well since $1 + 7 = 3 + 5 = 1/2CT$. Subsequently, the chosen instances were made up of parts with removal times of 3, 5, 7 and 11 and $CT = 26$. As a result, the optimal balance for all subsequent instances would consist of a perfect balance of precedence-preserving combinations of 3, 5, 7 and 11 at each workstation with idle times of 0. To further complicate the data (i.e., provide a large, feasible search space) only one part was listed as hazardous and this was one of the parts with the largest part removal time (the last one listed in the original data). In addition, exactly one part (the last listed, second largest part removal time component) was listed as being demanded. This was done so that only the hazardous and the demand sequencing would be demonstrated, while providing a slight

solution sequence disadvantage to any purely greedy methodology (since in an optimal solution, each workstation requires two parts with part removal times of 3 and 5 along with the larger part removal time parts, assigning hazardous and demanded parts to those smaller part removal time parts may allow some methodologies to artificially obtain the an optimal initial workstation; e.g., $PS_k = \langle 3, 5, 11, 7 \rangle$ would be easily obtained in the first workstation by a greedy methodology if the first part in that solution had been listed as hazardous, the second as demanded, and the third and forth selected on size). From each part removal time size, the first listed part was selected to have a removal direction differing from the other parts having the same part removal time. This was done to demonstrate direction selection while requiring DLBP GA to move these first parts of each part removal time size encountered to the end of the sequence (i.e., into the last workstation) in order to obtain the optimal direction value of $R^* = 1$ (assuming the solution technique being evaluated is able to successfully place the hazardous and demanded parts towards the front of the sequence). Also, there were no precedence constraints placed on the sequence, a deletion that further challenges any method's ability to attain an optimal solution.

The final configuration of the developed benchmark was: 19 instances with instance size evenly distributed from $n = 8$ to $n = 80$ in steps of $|PRT|$ (i.e., 4). This provided numerous instances of pre-determined, calculable solutions with the largest instance 10 times larger than the smallest instance. The size and range of the instances is considered appropriate and significant for this study, with small ns tested—which decreases the NWS and tends to exaggerate less than optimal performance—as well as large, which demonstrates time complexity growth and efficacy changes with n . To summarize, the a priori data used in this study consisted of $8 \leq n \leq 80$ parts with 4 unique part removal times giving $PRT = \{3, 5, 7, 11\}$. Only the last part having a part removal time of 11 is hazardous; only the last part having a part removal time of 7 is demanded. The first of each part with part removal times equal to 3, 5, 7 and 11 are removed in direction $+x$; all others are in

direction $-x$. The disassembly line is operated at a speed that allows 26 seconds ($CT = 26$) for each workstation. Known optimal results include $F^* = 0, H^* = 1, D^* = 2, R^* = 1$.

In general, for any n parts consisting of this type of data the following can be calculated:

$$NWS^* = \frac{n}{|PRT|}, \quad (30)$$

$$NWS_{nom} = n, \quad (31)$$

$$I^* = 0, \quad (32)$$

$$I_{nom} = \frac{n \cdot CT \cdot (|PRT| - 1)}{|PRT|}, \quad (33)$$

$$F^* = 0, \quad (34)$$

with F_{nom} given by Formula (11). Hazard measures are given by Formula (13) and values are assigned as

$$h_k = \begin{cases} 1, & k = n, \\ 0, & \text{otherwise,} \end{cases} \quad (35)$$

with demand measures are given by Formula (19) and values assigned as

$$d_k = \begin{cases} 1, & k = \frac{n \cdot (|PRT| - 1)}{|PRT|}, \\ 0, & \text{otherwise.} \end{cases} \quad (36)$$

Part removal direction measures are given by Formula (24) with values assigned as

$$r_k = \begin{cases} 1, & k = 1, \frac{n}{|PRT|} + 1, \frac{2n}{|PRT|} + 1, \dots, \frac{(|PRT| - 1) \cdot n}{|PRT|} + 1, \\ 0, & \text{otherwise.} \end{cases} \quad (37)$$

Since $|PRT| = 4$ in this paper, each part removal time is generated by

$$PRT_k = \begin{cases} 3, & 0 < k \leq \frac{n}{4}, \\ 5, & \frac{n}{4} < k \leq \frac{n}{2}, \\ 7, & \frac{n}{2} < k \leq \frac{3n}{4}, \\ 11, & \frac{3n}{4} < k \leq n. \end{cases} \quad (38)$$

7. Exhaustive search model description and the algorithm

An exhaustive search algorithm was developed to confirm the exponential time growth of exhaus-

tive search, to provide a time benchmark for the subsequent genetic algorithm study, and to determine the optimal solutions for any data set up to a given size (based on a reasonable search time). The exhaustive algorithm was built around a recursive function (Fig. 1) that generated all permutations of a sequence of n numbers with each number representing a disassembly task and the order of the numbers representing the disassembly sequence. The purpose of this exhaustive algorithm is to find every subset of the data set (every permutation) and from all those subsets, find the solution set that best satisfies the performance (balancing) requirements by checking against all other permutations. The solution performance was computed based on Formula (4) (comparison of equal F solutions for improved H , D and R measures is performed in a subsequent subroutine and not shown in Fig. 1).

```

PROCEDURE_GENERATE (q){
  IF ((q == 0) ^ (PROCEDURE_PRECEDENCE_PASS)){
    F = PROCEDURE_CALC_FITNESS;
    IF (FIRST_PERMUTATION_GENERATED){
      F* = F;
    }
    IF (F < F*){
      F* = F;
    }
  }
  FOR ∇ k ∈ P{
    PROCEDURE_EXCHANGE (k, q - 1);
    PROCEDURE_GENERATE (q - 1);
    PROCEDURE_EXCHANGE (k, q - 1);
  }
  RETURN;
}

PROCEDURE_EXCHANGE (a, b){
  temp = PSa;
  PSa = PSb;
  PSb = temp;
  RETURN;
}
    
```

Fig. 1. Recursive backtracking algorithm to generate all permutations of P numbers.

8. Exhaustive search algorithm numerical results

The exhaustive search algorithm and DLBP GA were both written in C++ and run on a 1.6 GHz PM x86-family workstation. The developed algorithms were investigated on a variety of test cases for verification and validation purposes. Fig. 2 provides the plot of runtime versus instance size for the exhaustive algorithm and the genetic algorithm, demonstrating how the exhaustive search runtime rapidly increases with instance size. With an instance size of less than 10, the problem can be solved within 1 minute. The time to complete the exhaustive search with $n = 12$ tasks took just over 20 minutes. At this rate, just 19 tasks would require almost 10,000 years. Faster computers and programming tricks can speed this up but the growth in exhaustive search is exponential nonetheless. Using the a priori instances in the range of $8 \leq n \leq 80$, only $n \leq 12$ sized instances were found to be quickly solved using the recursive exhaustive search algorithm. The algorithm's time complexity was shown to increase with instance size at about $2n!$.

9. Genetic algorithm model description and the algorithm

A genetic algorithm (a parallel neighborhood, stochastic directed search technique) provides an

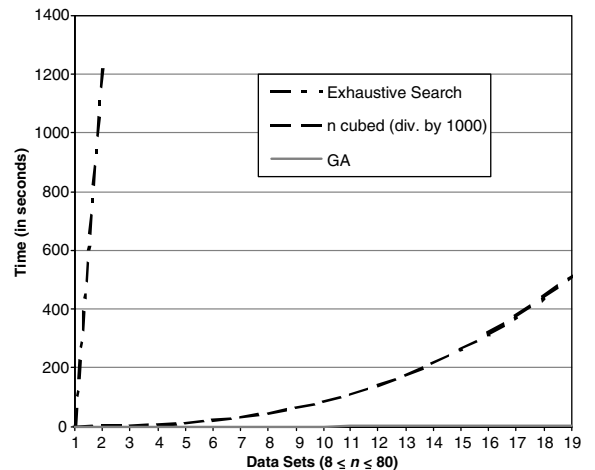


Fig. 2. Comparison of DLBP GA, $O(n^3)$ and exhaustive search time complexities.

environment where solutions continuously cross-breed, mutate and compete with each other until they evolve into the best solution. Due to its structure and search method, a GA is often able to find a global solution unlike many other heuristics that use hill climbing to find a best solution nearby, resulting only in a local optima. In addition, a GA does not need specific details about a problem nor is the problem's structure relevant; a function can be linear, nonlinear, stochastic, combinatorial, noisy, etc.

GA has a solution structure defined as a chromosome, which is made up of genes and generated by two parent chromosomes from the pool of solutions, each having its own measure of fitness. New solutions are generated from old using the techniques of crossover (sever parents genes and swap severed sections) and mutation (randomly vary genes within a chromosome). The main challenge with any genetic algorithm implementation is determining a chromosome representation that remains valid after each generation.

For DLBP, the chromosome (solution) consisted of a sequence of genes (parts). For example, if a chromosome in the pool consisted of the n -tuple $\langle 5, 2, 8, 1, 4, 7, 6, 3 \rangle$, then part number 5 would be removed first, followed by 2, then 8, etc. A population, or pool, of size N was used. Only feasible disassembly sequences were considered as members of the population or as offspring. The balance (F_i) and the number of workstations (NWS_i) were computed for each chromosome (Fig. 3) using the same method for solution performance determination as in the exhaustive search. Since measure of balance is the primary consideration in this paper, additional objectives are only considered subsequently; i.e., solutions having equivalent measures of balance are then compared for selection based on hazardous part placement and calculated using Formula (13); solutions also having equivalent hazard measures are compared for selection based on early removal of high demand parts, which is calculated using Formula (19); solutions also possessing equivalent demand measures are compared for selection based on the number of part removal direction changes calculated by Formula (24).

The GA for DLBP was constructed as follows. An initial, feasible population was randomly gen-

```

PROCEDURE_CALC_FITNESS{
  FOR  $\forall i \in N$ {
     $F_i = 0$ ;
     $I_i = 0$ ;
     $j = 0$ ;
     $ST_j = 0$ ;
    FOR  $\forall k \in G$ {
      IF ( $ST_j + PRT_{ik} > CT$ ){
         $F_i = F_i + (CT - ST_j)^2$ ;
         $I_i = I_i + (CT - ST_j)$ ;
         $j = j + 1$ ;
         $ST_j = 0$ ;
      }
       $ST_j = ST_j + PRT_{ik}$ ;
    }
     $F_i = F_i + (CT - ST_j)^2$ ;
     $I_i = I_i + (CT - ST_j)$ ;
     $NWS_i = j + 1$ ;
  }
}

```

Fig. 3. Fitness calculation algorithm to determine total number of workstations and overall fitness.

erated and the fitness of each chromosome in this generation was calculated. An even integer of $R_x \cdot N$ parents was randomly selected for crossover to produce $R_x \cdot N$ offspring (offspring make up $(R_x \cdot N \cdot 100)$ percent of each generation's population). An elegant crossover, the precedence preservative crossover (PPX) developed by Bierwirth et al. (1996), was used to create the offspring. As shown in Fig. 4, PPX first creates a mask (one for each child, every generation). The mask consists of random 1s and 2s indicating which parent part information should be taken from. If, for example, the mask for child 1 reads 221211, the first two parts (i.e., from left to right) in parent 2 would make up the first two genes of child 1 (and these parts would be stricken from the parts available to take from both parent 1 and 2); the first available (i.e., not stricken) part in parent 1 would make up gene three of child 1; the next available part in parent 2 would make up gene four of child 1; the last two parts in parent 1 would make up genes five and six of child 1. This technique is repeated using a new mask for child 2.

Parent 1:	1 3 2 6 5 8 7 4
Parent 2:	1 2 3 5 6 8 7 4
<u>Mask:</u>	<u>2 2 1 2 1 1 1 2</u>
Child:	
Parent 1:	1 3 2 6 5 8 7 4
Parent 2:	1 2 3 5 6 8 7 4
<u>Mask:</u>	<u>2 2 1 2 1 1 1 2</u>
Child:	1 2
Parent 1:	x 3 x 6 5 8 7 4
Parent 2:	x x 3 5 6 8 7 4
<u>Mask:</u>	<u>1 2 1 1 1 2</u>
Child:	1 2 3
Parent 1:	x x x 6 5 8 7 4
Parent 2:	x x x 5 6 8 7 4
<u>Mask:</u>	<u>2 1 1 1 2</u>
Child:	1 2 3 5
Parent 1:	x x x 6 x 8 7 4
Parent 2:	x x x x 6 8 7 4
<u>Mask:</u>	<u>1 1 1 2</u>
Child:	1 2 3 5 6 8 7
Parent 1:	x x x x x x x 4
Parent 2:	x x x x x x x 4
<u>Mask:</u>	<u>2</u>
Child:	1 2 3 5 6 8 7 4

Fig. 4. PPX example.

After crossover, mutation is randomly conducted. Mutation was occasionally (based on the R_m value) performed by randomly selecting a single child then exchanging two of its disassembly tasks while ensuring precedence is preserved. The $R_x \cdot N$ least fit parents are removed by sorting the entire parent population from worst-to-best based on fitness. Since the GA saves the best parents from generation to generation and it is possible for duplicates of a solution to be formed using PPX, the solution set could contain multiple copies of

the same answer resulting in the algorithm potentially becoming trapped in a local optima. This becomes more likely in a GA with solution constraints (such as precedence requirements) and small populations, both of which are seen in the study in this paper. To avoid this, DLBP GA was modified to treat duplicate solutions as if they had the worst fitness performance (highest numerical value), relegating them to replacement in the next generation. With this new ordering, the best unique $(1 - R_x) \cdot N$ parents were kept along with all of the $R_x \cdot N$ offspring to make up the next generation then the process was repeated. To again avoid becoming trapped in local optima, DLBP GA, as with many combinatorial optimization techniques, was run not until a desired level of performance was reached but rather for as long as deemed acceptable by the user. Since DLBP GA always keeps the best solutions from generation to generation, there is no risk of solution drift or bias and the possibility of mutation allows for a diverse range of possible solution space visits over time.

DLBP GA was modified from a general GA in several ways. Instead of the worst portion of the population being selected for crossover, in the DLBP GA all of the population was (randomly) considered for crossover. This better enables the selection of nearby solutions (i.e., solutions similar to the best solutions to date) common in many scheduling problems. Also, mutation was performed only on the children, not the worst parents. This was done to address the small population used in DLBP GA and to counter PPX's tendency to duplicate parents. A small population was used (20 versus the more typical 10,000 to 100,000) to minimize data storage requirements and simplify analysis (except for the case study where the generations were varied) while a large number of generations were used (10,000 versus the more typical 10–1000) to compensate for this small population while not being so large as to take an excessive amount of processing time. This was also done to avoid solving all cases to optimality since it was desirable to determine the point at which the DLBP GA performance begins to break down and how that breakdown manifests itself. Lower than the recommended 90% (Koza, 1992), a 60%

crossover was selected based on test and analysis. Sixty percent crossover provided better solutions and did so with one-third less processing time. Previous assembly line balancing literature indicating best results typically being found with crossover rates of from 0.5 to 0.7 also substantiated the selection of this lower crossover rate. A mutation was performed about one percent of the time. Although some texts recommend 0.01% mutation while applications in journal papers have used as much as 100% mutation, it was found that 1.0% gave excellent algorithm performance in our disassembly line balancing problem. Finally, duplicate children are sorted to make their deletion from the population likely since there is a tendency for the creation of duplicate solutions (due to PPX) and due to the small population saved from generation to generation.

10. Genetic algorithm case study numerical results

The developed GA was investigated on a variety of test cases to confirm its performance and to optimize parameters. An example from the literature was then selected as an application to an actual disassembly line balancing problem instance (Gungor and Gupta, 2002). This practical and relevant example from the literature consists of the data for the disassembly of a personal computer (PC) as shown in Table 1. It consists of 8 subassemblies (the number of genes is therefore 8) with part removal times of $PRT_k = (14, 10, 12, 18, 23, 16, 20, 36)$. The disassembly line operates at a speed that allows 40 seconds ($CT = 40$) for each

workstation. Since one purpose of this exercise was to provide analysis of DLBP GA performance with changes in generation size, the data was run varying the generations and without the use of hazard, demand or direction data to avoid potential complications. Also, due to GA's stochastic nature, this and all subsequent data were run a minimum of five times with the results averaged to avoid reporting unusually favorable or unusually poor results.

The GA converged to moderately good solutions very quickly. It was able to find at least one of the four solutions optimal in F (Table 2) after no more than 10 generations. 100 generations consistently provided three to four of the F -optimal solutions, while 1000 generations almost always resulted in the generation of all four optimal solutions. Similarly, the speed for the C++ implemented program searching 1000 generations of 20 chromosomes each was just over one second on the previously described workstation.

The balancing aspect of the algorithm worked extremely well with four equivalent (and, in fact, optimal) solutions being found. All four best solutions consisted of a total of four workstations with 2–4 seconds per workstation of idle time. Therefore, the obtained solutions were optimal in number of workstations and also provided idle times at each workstation of at least 5% but not more than 10% of the total disassembly time allocated to each workstation of 40 seconds. DLBP GA consistently and rapidly found all of the optimally balanced solutions in what approaches an exponentially large search space (potentially as large as $8!$ or 40,320).

11. Genetic algorithm benchmark data sets' numerical results

The developed GA was then used on the a priori benchmark test cases to further measure its performance and determine its limitations. Using the previously described population size and mutation rates and run for 10,000 generations using crossover rates of between 60% and 90% (with 60% reported here due to its more favorable time performance and efficacy), the DLBP GA's time

Table 1
Knowledge base of the personal computer example

Task	Part removal description	Time
1	PC top cover	14
2	Floppy drive	10
3	Hard drive	12
4	Back plane	18
5	PCI cards	23
6	RAM modules (2)	16
7	Power supply	20
8	Motherboard	36

Table 2
The four disassembly sequence solutions optimal in *F*

		Workstation				
		1	2	3	4	
<i>(a)</i>						
Part removal sequence	1	14				Time to remove part (in seconds)
	5	23				
	3		12			
	6		16			
	2		10			
	8			36		
	7				20	
	4				18	
Total time		37	38	36	38	
Idle time		3	2	4	2	
<i>(b)</i>						
Part removal sequence	1	14				Time to remove part (in seconds)
	5	23				
	3		12			
	2		10			
	6		16			
	8			36		
	7				20	
	4				18	
Total time	37	38	36	38		
Idle time	3	2	4	2		
<i>(c)</i>						
Part removal sequence	1	14				Time to remove part (in seconds)
	5	23				
	2		10			
	6		16			
	3		12			
	8			36		
	7				20	
	4				18	
Total time	37	38	36	38		
Idle time	3	2	4	2		
<i>(d)</i>						
Part removal sequence	1	14				Time to remove part (in seconds)
	5	23				
	2		10			
	3		12			
	6		16			
	8			36		
	7				20	
	4				18	
Total time	37	38	36	38		
Idle time	3	2	4	2		

complexity increased with instance size at about $0.1n$, see Fig. 5. Formally, we then list the time complexity of DLBP GA as $O(n)$. Although larger instances can be expected to take longer to solve with DLBP GA, it is a rather shallow, linear increase with significantly slower growth than the exhaustive search algorithm. The population size and number of generations was intentionally selected to allow for performance decreases; efficacy was shown to slowly drop off with instance size. An instance size of $n = 16$ was seen to be the point at which the optimally balanced solution was not consistently found for the selected N and number of generations. Although DLBP GA's performance decreased with instance size, it can be seen in Fig. 6 that the solution found, while not optimal, was consistently within a few percent of optimal. Fig. 7 allows a detailed study of the decrease in performance with increased instance size, demonstrating optimal performance until $n = 12$ with a rapid decrease in performance through $n = 16$, followed by minor performance improvements until $n = 60$ with a similar steep decrease in performance through $n = 72$, followed by another period of minor performance improvements. Also, the number of workstations required by DLBP GA typically stayed within 4% of the optimal (minimum) number of workstations required and was never seen to be worse than 7% (Fig. 8). DLBP GA maintained the

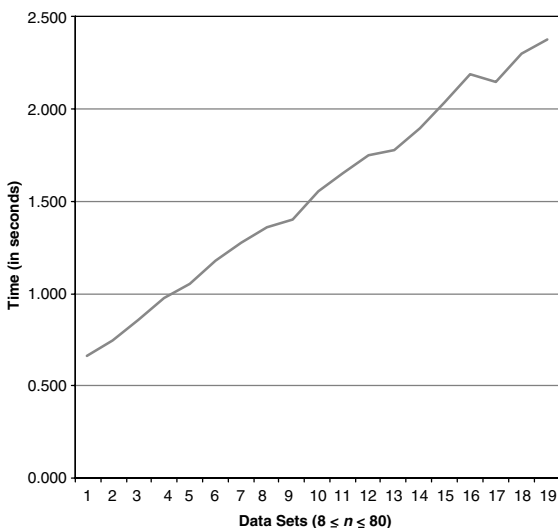


Fig. 5. Detailed time complexity of DLBP GA.

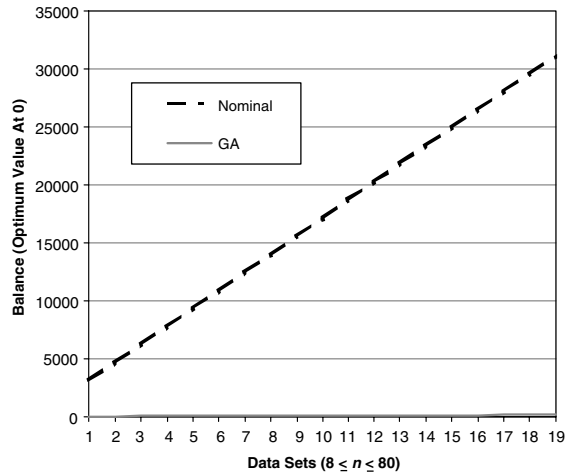


Fig. 6. Comparison of DLBP GA balance performance to best-case (at zero) and worst-case.

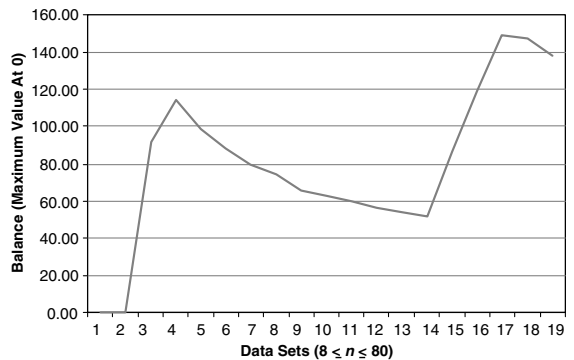


Fig. 7. Detailed GA balance performance.

measure of balance within 2% of optimum throughout. Since hazardous part removal (Fig. 9), early removal of high-demand parts (Fig. 10) and minimizing part removal directions (Fig. 11) were all subordinate to balance and to the precedence constraints (and to each other respectively), the performance of each of these was seen to degrade at rates proportionate to their priority; hazardous and high-demand part removal measures stayed within 37% and 45% of optimum respectively while the part removal direction measure varied from 15% of optimum to worst-case.

Although the preceding balance results are sub-optimal, this is more a reflection of the especially

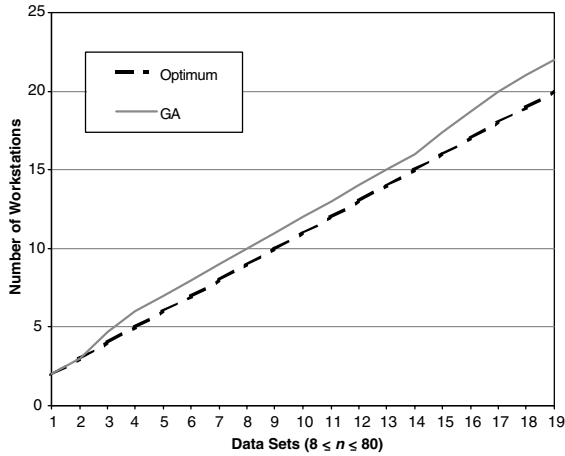


Fig. 8. DLBP GA workstation determination performance.

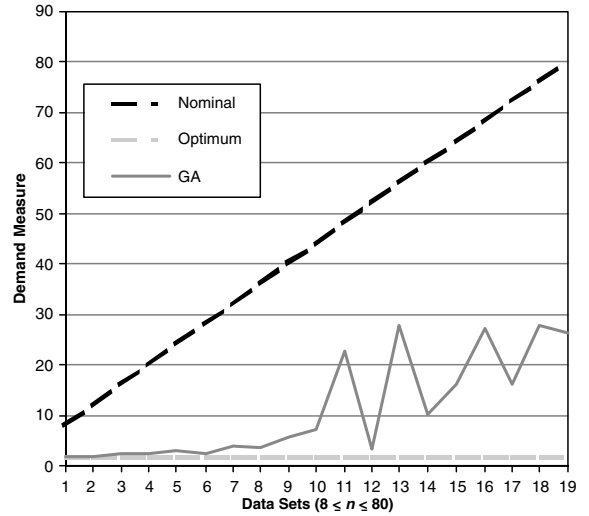


Fig. 10. Comparison of DLBP GA demand performance to best-case and worst-case.

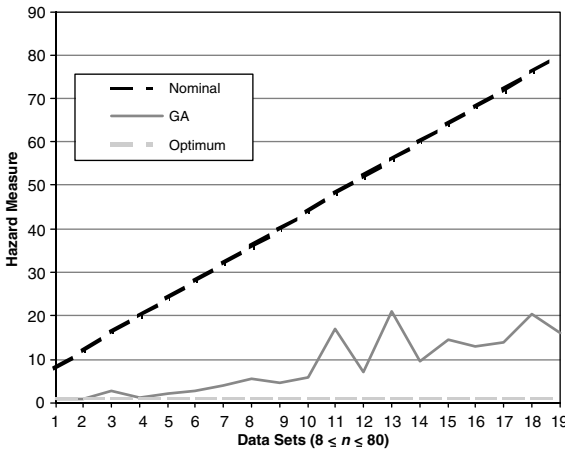


Fig. 9. Comparison of DLBP GA hazard performance to best-case and worst-case.

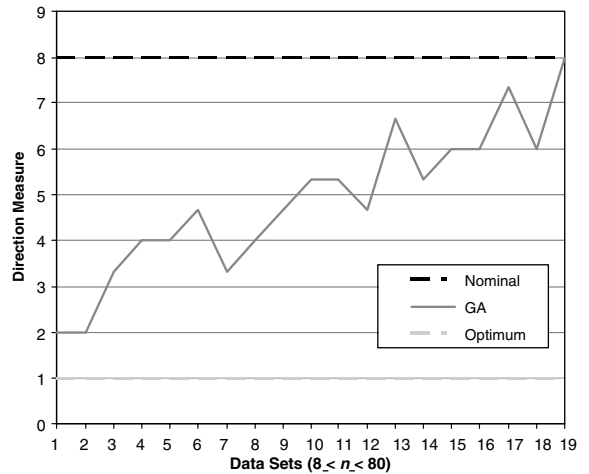


Fig. 11. Comparison of DLBP GA part removal direction performance to best-case and worst-case.

designed a priori data set than any search techniques. Sub-optimal solutions are not an atypical result when this data set is evaluated. The data has been designed to pose challenges—even at relatively small n values—to a variety of combinatoric solution-generating techniques.

Note that the inclusion of precedence constraints will increasingly move DLBP GA towards the optimal solution. F , H , D and R performance can also be improved through the use of a larger population and more generations, while time complexity improvements can be gained with smaller generations and smaller populations, though each

of these improvements requires a tradeoff in the other.

12. Conclusions

Although a near-optimum combinatorial optimization technique, the DLBP version of the GA metaheuristic quickly found optimally balanced

solutions, or solutions within single digit percentages of optimal, in a variety of exponentially large search spaces, with the level of optimality increasing with the number of constraints. It was able to generate a feasible sequence with an optimal or near-optimal measure of balance while maintaining or improving the hazardous materials, part demand and part removal direction measures. The proposed balancing method worked remarkably well throughout, while the developed a priori data set instances provided a valuable tool for evaluating the DLBP, providing data with a calculable solution in the intractably large search space. DLBP GA appears to be well suited to the multi-criteria decision making problem format. Also, DLBP GA can easily be structured for multi-processing on a network of computers for order-of-magnitude increases in processing speed. In addition, the GA is ideally suited to integer problems, a requirement of many disassembly problems, which generally do not lend themselves to rapid or easy solution by traditional optimum solution generating mathematical programming techniques.

References

- Bierwirth, C., Mattfeld, D.C., Kopfer, H., 1996. On permutation representations for scheduling problems, parallel problem solving from nature. In: Voigt, H.M., Ebeling, I., Rechenberg, I., Schwefel, H.P. (Eds.), *Lecture Notes in Computer Science*, 1141. Springer-Verlag, Berlin, Germany, pp. 3.10–3.18.
- Brennan, L., Gupta, S.M., Taleb, K.N., 1994. Operations planning issues in an assembly/disassembly environment. *International Journal of Operations and Production Planning* 14 (9), 57–67.
- Elsayed, E.A., Boucher, T.O., 1994. *Analysis and Control of Production Systems*. Prentice Hall, Upper Saddle River, NJ.
- Erel, E., Gokcen, H., 1964. Shortest-route formulation of mixed-model assembly line balancing problem. *Management Science* 11 (2), 308–315.
- Garey, M., Johnson, D., 1979. *Computers and Intractability: A Guide to the Theory of NP Completeness*. W.H. Freeman and Company, San Francisco, CA.
- Gungor, A., Gupta, S.M., 1999. Issues in environmentally conscious manufacturing and product recovery: A survey. *Computers and Industrial Engineering* 36 (4), 811–853.
- Gungor, A., Gupta, S.M., 2001. A solution approach to the disassembly line problem in the presence of task failures. *International Journal of Production Research* 39 (7), 1427–1467.
- Gungor, A., Gupta, S.M., 2002. Disassembly line in product recovery. *International Journal of Production Research* 40 (11), 2569–2589.
- Gupta, S.M., Taleb, K.N., 1994. Scheduling disassembly. *International Journal of Production Research* 32, 1857–1866.
- Gutjahr, A.L., Nemhauser, G.L., 1964. An algorithm for the line balancing problem. *Management Science* 11 (2), 308–315.
- Koza, J.R., 1992. *Genetic Programming: On the Programming of Computers by the Means of Natural Selection*. MIT Press, Cambridge, MA.
- Lambert, A.J.D., Gupta, S.M., 2005. *Disassembly Modeling for Assembly, Maintenance, Reuse, and Recycling*. CRC Press, Boca Raton, FL.
- McGovern, S.M., Gupta, S.M., 2004. Combinatorial optimization methods for disassembly line balancing. In: *Proceedings of the 2004 SPIE International Conference on Environmentally Conscious Manufacturing IV*, Philadelphia, Pennsylvania, October 25–28, pp. 53–66.
- McGovern, S.M., Gupta, S.M., Kamarthi, S.V., 2003. Solving disassembly sequence planning problems using combinatorial optimization. In: *Proceedings of Northeast Decision Sciences Institute 32nd Annual Meeting*, pp. 178–180.
- Suresh, G., Vinod, V.V., Sahu, S., 1996. A genetic algorithm for assembly line balancing. *Production Planning and Control* 7 (1), 38–46.