January 01, 2004

Combinatorial optimization methods for disassembly line balancing

Seamus M. McGovern
Northeastern University

Surendra M. Gupta
Northeastern University

Laboratory for Responsible Manufacturing

Bibliographic Information

McGovern, S. M. and Gupta, S. M., "Combinatorial Optimization Methods for Disassembly Line Balancing", ***Proceedings of the SPIE International Conference on Environmentally Conscious Manufacturing IV***, Philadelphia, Pennsylvania, pp. 53-66, October 26-27, 2004.

Copyright Information

Contact Information

Dr. Surendra M. Gupta, P.E.
Professor of Mechanical and Industrial Engineering and
Director of Laboratory for Responsible Manufacturing
334 SN, Department of MIE
Northeastern University
360 Huntington Avenue
Boston, MA 02115, U.S.A.

(617)-373-4846 ***Phone***
(617)-373-2921 ***Fax***
gupta@neu.edu ***e-mail address***

http://www.coe.neu.edu/~smgupta/ ***Home Page***

Combinatorial optimization methods for disassembly line balancing

Seamus M. McGovern and Surendra M. Gupta*
Laboratory for Responsible Manufacturing
334 SN, Department of MIE
Northeastern University
360 Huntington Avenue
Boston, MA, 02115 USA

ABSTRACT

Disassembly takes place in remanufacturing, recycling, and disposal with a line being the best choice for automation. The disassembly line balancing problem seeks a sequence which: minimizes workstations, ensures similar idle times, and is feasible. Finding the optimal balance is computationally intensive due to factorial growth. Combinatorial optimization methods hold promise for providing solutions to the disassembly line balancing problem, which is proven to belong to the class of NP-complete problems. Ant colony optimization, genetic algorithm, and H-K metaheuristics are presented and compared along with a greedy/hill-climbing heuristic hybrid. A numerical study is performed to illustrate the implementation and compare performance. Conclusions drawn include the consistent generation of optimal or near-optimal solutions, the ability to preserve precedence, the speed of the techniques, and their practicality due to ease of implementation.

Keywords: Disassembly, disassembly line balancing, combinatorial optimization, H-K metaheuristic, ant colony optimization, genetic algorithm, greedy search, AEHC, product recovery

1. INTRODUCTION

More and more manufacturers are working to recycle and remanufacture their post-consumed products due to new and more rigid environmental legislation, increased public awareness, and extended manufacturer responsibility. In addition, the economic attractiveness of reusing products, subassemblies and parts instead of disposing of them has further fueled this effort. Recycling is 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 conditions. Thus, remanufacturing provides the quality standards of new products with used parts.

In order to minimize the amount of waste sent to landfills, product recovery seeks to obtain materials and parts from old or outdated products through recycling and remanufacturing – this includes the 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 defined as the methodical extraction of valuable parts/subassemblies and materials from discarded products through a series of operations. After disassembly, reusable components are cleaned, refurbished, tested and directed to inventory for remanufacturing operations. The recyclable materials can be sold to raw-material suppliers, while the residuals are sent to landfills.

Recently, disassembly has 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 conditions of the products received are 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

* gupta@neu.edu; phone 1 617 373-4846; fax 1 617 373-2921; URL http://www.coe.neu.edu/~smgupta/

Environmentally Conscious Manufacturing IV, edited by Surendra M. Gupta,
Proceedings of SPIE Vol. 5583 (SPIE, Bellingham, WA, 2004)
0277-786X/04/$15 · doi: 10.1117/12.570493

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 disassembly line balancing. The reusability of parts creates a demand for these parts, however, the demands and availability of the reusable parts is significantly less predicable than what is found in the assembly process. Finally, 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.

This paper first defines the disassembly line balancing problem (DLBP) then proves for the first time that it belongs to the class of NP-complete problems, necessitating specialized solution techniques. Combinatorial optimization is an emerging field that combines techniques from applied mathematics, operations research and computer science to solve optimization problems over discrete structures. These techniques include: greedy algorithms, integer and linear programming, branch-and-bound, divide and conquer, dynamic programming, local optimization, simulated annealing, genetic algorithms, and approximation algorithms. In this paper, the DLBP is solved using several combinatorial optimization methods. These include ant colony optimization (ACO), genetic algorithm (GA), and Hunter-Killer (H-K) metaheuristics, as well as a hybrid process consisting of a greedy sorting algorithm followed by a hill-climbing heuristic (Adjacent Element Hill Climbing; AEHC). While exhaustive search consistently provides the problem's optimal solution, its time complexity quickly limits its practicality. The application of these techniques is instrumental in rapidly obtaining solutions to the DLBPs intractably large solution space. The ACO considered here is an ant system algorithm known as the ant-cycle model [[4]]. The GA involves a randomly generated initial population with crossover and mutation performed over many generations. A DLBP implementation of the H-K metaheuristic – a deterministic search technique based upon an enhanced exhaustive search – is then applied and compared. The greedy algorithm considered here is based on the First-Fit-Decreasing (FFD) algorithm (developed for the bin-backing problem and effectively used in computer processor scheduling). AEHC is then used to ensure that the idle times at each workstation are similar. AEHC only compares tasks assigned in adjacent workstations; this is done both to conserve search time and to only investigate swapping tasks that will most likely result in a feasible sequence. All techniques seek to preserve precedence relationships. Examples are considered to illustrate the implementation of the methodologies. The conclusions drawn from the study include the consistent generation of optimal or near-optimal solutions, the ability to preserve precedence relationships, the speed of the methods, and their practicality due to the ease of implementation in solving the DLBP.

2. LITERATURE REVIEW

Product recovery involves a number of steps [[8]]. The first crucial step is disassembly. Disassembly is a methodical extraction of valuable parts/subassemblies and materials from post-used products through a series of operations [[2]], [[14]]. After disassembly, re-usable 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 and the residuals are disposed of.

Gungor and Gupta presented the first introduction to the disassembly line-balancing problem [[9]], [[10]], [[12]] 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 [[11]]. McGovern and Gupta developed a greedy/2-Opt hybrid algorithm for the DLBP [[18]]. Kongar and Gupta applied a GA to the disassembly sequencing problem [[16]]. McGovern *et al*. first applied combinatorial optimization techniques to the DLBP [[23]].

3. NOTATION

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

$<\cdot$	partial order; i.e., *x* precedes *y* is written $x <\cdot y$
α	weight of existing pheromone (trail) in path selection
β	weight of the edges in path selection
$\Delta\psi_k$	k^{th} element's delta skip measure; difference between problem size, *n*, and skip size, ψ_k (i.e., for $\Delta\psi = 10$ and $n = 80$, $\psi = 70$)
η_{pq}	ACO visibility value of edge *pq* (directed edge for DLBP)

ρ	variable such that $1 - \rho$ represents the pheromone evaporation rate
$\tau_{pq}(NC)$	amount of trail on directed edge *pq* during cycle *NC* (directed edge for DLBP)
ψ_k	k^{th} element's skip measure (i.e., for the solution's third element, visit every 2nd possible task if $\psi_3 = 2$)
c	initial amount of pheromone on each of the paths
CT	cycle time; maximum time available at each workstation
F	McGovern-Gupta measure of balance for a given solution sequence
F^*	optimal McGovern-Gupta measure of balance
F_{nr}	McGovern-Gupta measure of balance of ant *r* sequence at time *n* (the end of a tour)
F_{tr}	McGovern-Gupta measure of balance of ant *r* sequence at time *t*
I	total idle time for a given solution sequence
I^*	optimum (minimum) total idle time
I_{nom}	worst-case (maximum) total idle time
ISS_k	binary value; 1 if k^{th} part is in solution sequence, else 0
j	workstation count (1,…, *NWS*)
k	part identification (1,…, *n*)
L_r	ACO delta trail divisor value; here set equal to F_{nr}
m	number of ants
n	number of parts for removal
N	the set of natural numbers (0, 1, 2, …)
N	GA population size
NC_{max}	maximum number of cycles for ACO
NWS	number of workstations required for a given solution sequence
NWS^*	optimum (minimum) number of workstations
NWS_{nom}	worst-case (maximum) number of workstations
p	edge variable
P	the set of part removal tasks
$p^r_{pq}(t)$	probability of ant *r* taking an edge *pq* at time *t* (directed edge for DLBP)
PRT_k	part removal time required for *k*th part
$\|PRT_k\|$	cardinality of the set of part removal times
PS_k	k^{th} part in a solution sequence (i.e., for solution "3, 1, 2" $PS_2 = 1$)
q	edge variable
Q	amount of pheromone added if a path is selected
r	ant count
R_m	mutation rate
R_x	crossover rate
ST_j	station time; total processing time requirement in workstation *j*
t	time within a cycle; ranges from 0 to *n*
V	maximum variability for a given workstation idle time
x	counter variable
y	counter variable
Z^+	the set of positive integers (1, 2, …)
Z_p	p^{th} objective to minimize or maximize

4. THE DLBP MODEL DESCRIPTION

4.1. DLBP considered

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

1. minimize the number of disassembly workstations and hence, minimize the total idle time, and
2. ensure the idle times at each workstation are similar,

with a major constraint being to provide a feasible disassembly sequence for the product being investigated. The result is an integer, deterministic, decision making problem with an exponential search space. Testing a given solution against the

precedence constraints fulfills the major constraint of precedence preservation. Minimizing the sum of the workstation idle times also minimizes the total number of workstations and is described by:

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

which attains objective 1. This objective is represented as:

$$\text{Minimize } Z_1 = \sum_{j=1}^{NWS} (CT - ST_j) \tag{2}$$

Line balancing seeks to achieve Perfect Balance (all idle times equal to zero). When this is not achievable, either Line Efficiency (IE) or the Smoothness Index (SI) is used as a performance evaluation tool [[6]]. We use a measure of balance that combines the two and is easier to calculate. SI rewards similar idle times at each workstation, but at the expense of allowing for a large (suboptimal) number of workstations. This is because SI compares workstation elapsed times to the largest ST_j instead of the CT. IE rewards the minimum number of workstations, but allows unlimited variance in idle times between workstations because no comparison is made between ST_js. This paper makes use of the balancing method developed by McGovern and Gupta [[21]]. The McGovern-Gupta balancing method simultaneously minimizes the number of workstations while aggressively ensuring that idle times at each workstation are similar, though at the expense of the generation of a nonlinear objective function. 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. The McGovern-Gupta measure of balance is represented as:

$$F = \sum_{j=1}^{NWS} (CT - ST_j)^2 \tag{3}$$

with the DLBP balancing objective represented as:

$$\text{Minimize } Z_2 = \sum_{j=1}^{NWS} (CT - ST_j)^2 \tag{4}$$

Perfect Balance is indicated by:

$$Z_2 = 0 \tag{5}$$

Note that mathematically, formula (4) effectively makes formula (2) redundant due to the fact that it concurrently minimizes the number of workstations.

In addition, we find:

$$NWS^* = \left\lceil \frac{\sum_{k=1}^{n} PRT_k}{CT} \right\rceil \tag{6}$$

$$NWS_{nom} = n \tag{7}$$

4.2. Problem class

The decision version of DLBP is NP-complete (and hence, the optimization version is NP-hard). Using the concise style of Garey and Johnson [[7]], this is shown in the following proof by restriction (verification is omitted for brevity; DLBP is easily seen to be solvable in polynomial time by a nondeterministic algorithm).

INSTANCE: Set P of tasks, partial order $<\cdot$ on P, task time $PRT_k \in \mathrm{Z}^+$ for each $k \in P$, workstation capacity $CT \in \mathrm{Z}^+$, number $NWS \in \mathrm{Z}^+$ of workstations, difference between largest and smallest idle time $V \in \mathrm{N}$.

QUESTION: Is there a partition of P into disjoint sets $P_A, P_{B,\ldots,} P_{NWS}$ 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, and it obeys the precedence constraints?

Proof: Restrict to PARTITION by allowing only instances in which $CT = \left(\sum_{k=1}^{n} PRT_k \Big/ 2 \right) \in \mathrm{Z}^+$, $V = 0$ and $<\cdot$ is empty.

4.3. Problem assumptions

Problem assumptions include the following:

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

5. THE GREEDY AND AEHC HEURISTICS

A specially designed, two-phase hybrid approach is used to provide a very fast, near-optimal solution. The first phase provides a feasible solution to the DLBP and minimum or near-minimum *NWS* using a greedy algorithm. The second phase compensates for the algorithm's inability to balance the workstations by using AEHC.

5.1. Greedy model description and the algorithm

A greedy strategy always makes the choice that looks the best at the moment, that is, it makes a locally optimal choice in the hope that this choice will lead to a globally optimal solution. Greedy algorithms do not always yield optimal solutions, but for many problems they do [[3]]. The DLBP Greedy algorithm was built around FFD rules (FFD looks at each element in a list, from largest to smallest – *PRT*s in the DLBP – and puts that element into the first workstation in which it fits).

Each part in the sorted list is examined from first to last. If the part had not previously been put into the solution sequence (as described by ISS_k), the part is put into the current workstation if idle time remains to accommodate it and as long as putting it into the sequence at that position will not violate any of its precedence constraints. If no workstation can accommodate it at the given time in the search due to precedence constraints, the part is maintained on the sorted list (i.e., its ISS_k value remains 0) and the next part (not yet selected) on the sorted list is considered. If all parts have been examined for insertion into the current workstation, a new workstation is created and the process is repeated.

While being very fast and efficient, the algorithm is not always able to minimize the number of workstations. In addition, there is no capability to balance the workstations; in fact, the FFD structure lends itself to filling the earlier workstations as much as possible, often to capacity, while later workstations have progressively greater and greater idle times. This results in extremely poor balance. This limitation led to the development of AEHC to fulfill the second objective [[20]].

5.2. Hill-climbing description and the AEHC heuristic

Hill-climbing is an iterated improvement algorithm, basically a gradient descent/ascent. It makes use of an iterative greedy strategy, which is to move in the direction of increasing value. A hill-climbing algorithm evaluates the successor states and keeps only the best one [[15]]. AEHC is designed to consider swapping each task in every workstation with each task in the subsequent workstation in search of improved balance. It does this while observing precedence constraints and not exceeding *CT* in any workstation. Only adjacent workstations are compared to enable a rapid search and since it is deemed unlikely that parts several workstations apart can be swapped and still preserve the precedence of all of the tasks in-between. Specifically, AEHC goes through each part in each workstation and compares it to each part in the next adjacent workstation. If the two parts can be exchanged while preserving precedence, without exceeding

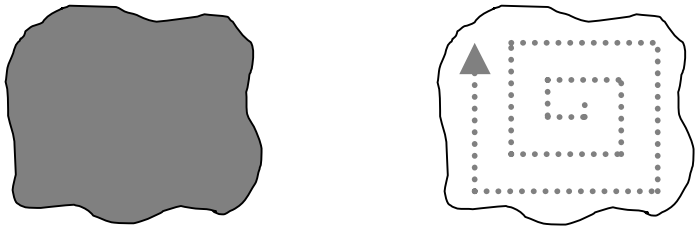
either workstations available idle time, and with a resulting improvement in the overall balance, the exchange is made and the resulting solution sequence is saved as the new solution sequence. This process is repeated until task elements of the last workstation have been examined. As is the norm with greedy algorithms, the DLBP Greedy process is run once to determine a solution. Hill-climbing, however, can be continuously run on subsequent solutions for as long as is deemed acceptable by the user or until it is no longer possible to improve, at which point it is assumed that the (local) optimum has been reached [[15]].

6. THE H-K METAHEURISTIC

A recently described search approach is applied next. This metaheuristic provides a near-optimal solution to combinatorial optimization problems using a modified exhaustive search technique that provides data sampling of all solutions.

6.1. H-K metaheuristic background and motivation

In some search applications (anti-submarine warfare, search and rescue), exhaustive search is not possible due to time or sensor limitations. In these cases, it becomes necessary to sample the search space and operate under the assumption that the best point found is either the optimal point or is reasonably near. First introduced by McGovern and Gupta [[19]], the H-K metaheuristic works by sampling the exhaustive solution set; it searches the solution space in a method similar to exhaustive search using a pattern that skips solutions to significantly reduce the search space (Figs. 1a and 1b).



Figs. 1a and 1b. Exhaustive and H-K search space.

Once the solution is generated, it can be further refined by performing subsequent local searches (such as 2-Opt or smaller, localized H-K searches; i.e., using the solution as the starting point, decreasing the skip size, or limiting the range of each element in the subsequent search). Depending on the application, H-K can be run once, multiple times from the same starting point using a different ψ, multiple times from a different starting point using the same ψ, or followed up with a differing local search on the best or on several of the top solutions generated. In this paper, H-K is run alone to a single-phase solution. Because H-K is exceptionally deterministic and performs no preprocessing of the data, the order in which the data is presented can be expected to affect the solutions generated. For this reason, the analyses done in this paper is run twice; first with the data as given by formula (17) and then again with the data presented in reverse order. The better of the two results are then maintained as the sole, best solution. Also, H-K is run here on multiple skip sizes (specifically, all skips that are of $n - 10$ and larger, i.e., $n-10 \le \psi \le n-1$) in addition to the data in forward and in reverse format. In this basic form, there are two processes with starting points of PS_k = (1, 1, 1, … , 1) (given PRT_k = (1, 2, 3, … , n); note that since the solution set is a permutation, there are no repeated items, therefore the starting point is effectively PS_k = (1, 2, 3, … , n)) and PS_k = (n, n, n, … , n) (effectively PS_k = (n, $n - 1$, $n - 2$, $n - 3$, … , 1)), constant skip type (i.e., each element in the solution sequence is skipped in the same way), varying skip size ($n-10 \le \psi \le n-1$), and no follow-on solution refinement.

6.2. H-K process and DLBP application

In the basic H-K and with ψ = 2, the first element in the first solution would be 1, the next element considered would be 1, but since it is already in the solution, that element would be incremented and 2 would be considered and be acceptable. This is repeated for all of the elements until the first solution is generated. It repeats in a nested fashion for all solution elements until the first element is the only one remaining to be incremented. In this next iteration, the first element under consideration would be incremented by 2 and therefore, 3 would be considered and inserted as the first element. Since 1 is not yet in the sequence, it would be placed in the second position, 2 in the third, etc. For the DLBP H-K this is further modified to test the proposed sequence part addition for precedence constraints. If all possible parts for a given solution position fail these checks, the remainder of the positions are not further inspected, the procedure falls back to the

previously successful solution element addition, increments it and continues. These processes are repeated until all allowed items have been visited in the first solution position (and by default, due to the nested nature of the search, all subsequent solution positions). For example, with n = 4 and $PRT_k \in$ {1, 2, 3, 4}, instead of considering the 4! = 24 possible permutations, only five are considered by the single-phase H-K with ψ = 2 and using forward-only data: PS_k = (1, 2, 3, 4), PS_k = (1, 4, 2, 3), PS_k = (3, 1, 2, 4), PS_k = (3, 1, 4, 2), and PS_k = (3, 4, 1, 2). Just as with DLBP Greedy/AEHC, all of the parts are maintained in the Tabu-type list, ISS_k. After all feasible DLBP H-K solutions are generated, it then places each element into a workstation using the Next-Fit rule. DLBP H-K looks at each element in the solution and puts that element into the current workstation if it fits; if it does not fit, a new workstation is assigned. When all of the work elements have been assigned to a workstation, the process is complete and the balance measure is calculated. The best of all of the inspected solution sequences is then saved as the problem solution.

7. THE ACO METAHEURISTIC

7.1. ACO model description

ACO is a probabilistic evolutionary algorithm based on a distributed autocatalytic process that makes use of agents called ants. Just as a colony of ants can find the shortest distance to a food source, in ACO they work cooperatively towards a solution. Ants are placed at multiple starting nodes, such as cities for the traveling salesman problem. Each of the m ants is allowed to visit all remaining unvisited edges as indicated by the previously described Tabu-type list. Each ant's possible subsequent steps (from some node p to a node q giving edge pq) are evaluated for desirability and each is assigned a proportionate probability given by formula (8). Based on these, the next edge is randomly selected for each ant. After completing an entire tour, the ant with the best solution is given the equivalent of additional pheromone (in proportion to tour desirability) that is added to each step that ant visited. All paths are then decreased in their pheromone strength according to a measure of evaporation. This process is repeated for NC_{max} or until stagnation behavior (where all ants make the same tour). The probability of ant r taking edge pq at time t is:

$$p_{pq}^{r}(t) = \begin{cases} \dfrac{[\tau_{pq}(t)]^{\alpha} \cdot [\eta_{pq}]^{\beta}}{\sum_{r \in allowed_r} [\tau_{pr}(t)]^{\alpha} \cdot [\eta_{pr}]^{\beta}} & \text{if} \quad q \in allowed_r \\ 0 & otherwise \end{cases} \tag{8}$$

Trail for each edge visited after each cycle is calculated using:

$$\tau_{pq}(NC+1) = \rho \cdot \tau_{pq}(NC) + \Delta\tau_{pq} \tag{9}$$

where:

$$\Delta\tau_{pq} = \sum_{r=1}^{m} \Delta\tau_{pq}^{r} \qquad \text{and:} \qquad \Delta\tau_{pq}^{r} = \begin{cases} \dfrac{Q}{L_r} & edge(p,q) used \\ 0 & otherwise \end{cases} \tag{10}$$

7.2. DLBP-specific modifications

DLBP ACO is modified to account for feasibility constraints and provide for multiple objectives [[13]]. In DLBP, a solution consists of a sequence of work elements (i.e., tasks, components, or parts). For example, if a sequence consisted of "5 2 8 1 4 7 6 3," then part 5 would be removed first, followed by part 2, then part 8, and so on. In DLBP ACO, each part is a node, with the number of ants initially being set equal to the number of parts, and one ant on each part. Each ant is allowed to visit all parts not already in its solution. Each ant's possible subsequent steps are evaluated for feasibility and the McGovern-Gupta measure of balance then assigned a proportionate probability. Infeasible steps receive a probability of 0 with those ants effectively ignored for the remainder of the cycle. At tour completion, the best solution found is saved and the process is repeated. For DLBP ACO, the visibility, η_{pq}, is defined as:

$$\eta_{pq} = \frac{1}{F_{tr}} \tag{11}$$

while in formula (10), L_r is set to F_{nr}, where a small final value for ant *r*'s measure of balance at time *n* (the end of the tour) provides a large measure of trail added to each edge. Though L_r and η_{pq} are related here, this is not unusual in ACO applications (in the traveling salesman problem, L_r is the tour length, while η_{pq} is the reciprocal of the distance between cities *p* and *q*). However, this method of selecting η_{pq} (effectively a short-term greedy choice) may not always translate into the best final solution for a complex problem like DLBP. Also, since all edges are directed edges in DLBP, *pq* is evaluated separately from *qp*. In addition, NC_{max} is set to 300 since larger problems than that studied here were shown to reach their best solution by that count. The process is not run until no improvements were shown but, as is the norm with many combinatorial optimization techniques, is run continuously until NC_{max} [[15]]. This also enabled the probabilistic component of ACO an opportunity to leave any local minima. Per the best ACO performance experimentally determined by Dorigo *et al.* [[5]], the following is used: $\alpha = 1.00$, $\beta = 5.00$, $\rho = 0.50$, Q = 100.00, and c = 1.00.

8. THE DLBP GENETIC ALGORITHM

8.1. GA model description

A GA has a solution structure defined as a chromosome, which is made up of genes (parts in DLBP) and generated by two parent chromosomes (each with its own measure of fitness based on the section 4 objectives) from the pool (population), *N*, of solutions. New solutions are made from old by the methods of crossover (sever parent genes and swap severed sections) and mutation (randomly vary genes within a chromosome). Only feasible disassembly sequences are considered as members of the population. The fitness is computed for each chromosome using formula (3). Initially, a feasible population is randomly generated and the fitness of each chromosome in this generation is calculated. An even integer of R_xN parents is randomly selected for crossover to produce R_xN offspring (offspring make up R_xN x 100% of each generation's population). A major challenge with any GA implementation is determining a chromosome representation that remains valid after each generation; the precedence preservative crossover (PPX) [[1]] was used here. 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 (from left to right) in parent 2 would make up the first two genes of child 1 and these parts would be stricken from those available to take from either parent 1 or 2. The first part (not stricken) 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. Mutation is randomly (based on the R_m value) performed by selecting a single child and exchanging two of its disassembly tasks (while ensuring precedence is preserved) after crossover. The R_xN least fit parents are removed by sorting the entire parent population (worst to best) and the process is repeated.

8.2. DLBP-specific modifications

DLBP GA was modified from a general-purpose GA in several ways. Instead of the worst portion of the population being selected for crossover, all of the population was randomly considered for crossover to enable more population diversity. Also, mutation was performed only on the children. This was done to address the small population used (since there is no desire to carry over a poor performing parent unchanged for generations) and to counter PPXs tendency to duplicate parents. Since DLBP GA saves the best parents from generation to generation and it is possible for solution duplicates (due to PPX), the population could contain multiple copies of the same solution resulting in the metaheuristic becoming trapped in a local optima. This becomes more likely with solution constraints (such as precedence requirements) and small populations, both of which are seen here. To avoid this, DLBP GA was modified to treat duplicate solutions as if they had the worst fitness performance, relegating them to replacement in the next generation. With this new ordering, the best unique $(1 - R_x)N$ parents were kept along with all of the R_xN offspring to make up the next generation. To again avoid becoming trapped in local optima, the DLBP GA was run, not until a desired level of performance was reached but (as in DLBP ACO) for as long as was acceptable by the user; since this 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. A small population was used (20 versus the more typical 10,000 to 100,000) to minimize data storage requirements and simplify analysis, while a large number of generations were used (10,000 versus the more typical 10 to 1,000) to compensate for this small population, while not being so large as to take an excessive amount of processing time. Lower than the recommended 90% [[17]], a 60% crossover was selected based on test and analysis (60% crossover provided better solutions and did so with 1/3 less

processing time). Previous assembly line balancing and disassembly GA literature indicate best results typically being found with crossover rates of 0.5 to 0.7, also substantiated the use of this lower crossover rate. Mutation was performed 1% of the time. Although some texts recommend 0.01% mutation and applications in journal papers have used as much as 100% mutation, it was found that 1.0% gave excellent performance in DLBP. Finally, duplicate children are sorted to make their deletion from the population likely.

9. NUMERICAL RESULTS

The developed combinatorial optimization techniques were investigated on a variety of test cases to confirm their performance and to optimize any parameters. The methods were then used on test cases to further demonstrate their performance as well as their limitations. This was done 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 $1 + 7 = 3 + 5 = \frac{1}{2}\, CT$. 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 giving idle times of 0. To further complicate the data (i.e., provide a large, feasible search space), there were no precedence constraints placed on the sequence, a deletion that further challenges any methods' 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 $|PRT_k|$ (i.e., 4). This provided numerous instances of predetermined, 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 *n*s 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 test data consisted of $8 \leq n \leq 80$ parts with 4 unique part removal times giving $PRT_k \in \{3, 5, 7, 11\}$. The disassembly line is operated at a speed that allows 26 seconds ($CT = 26$) for each workstation [[21]]. In general, for any *n* parts consisting of this *a priori* data, the following can be calculated:

$$NWS^{*} = \frac{n}{|PRT_k|} \tag{12}$$

$$NWS_{nom} = n \tag{13}$$

$$I^{*} = 0 \tag{14}$$

$$I_{nom} = \frac{nCT\ (|PRT_k| - 1)}{|PRT_k|} \tag{15}$$

$$F^{*} = 0 \tag{16}$$

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

$$PRT_k = \begin{cases} 3, & 0 < k \leq \frac{n}{4} \\ 5, & \frac{n}{4} < k \leq \frac{n}{2} \\ 7, & \frac{n}{2} < k \leq \frac{3n}{4} \\ 11, & \frac{3n}{4} < k \leq n \end{cases} \tag{17}$$

Note that a data set containing any parts with equal *PRT*s and no precedence constraints will have more than one optimal sequence. To properly gauge the performance of any solution-generating technique on the *a priori* data, the size of the set of optimal sequences needs to be quantified. From probability theory we know that, for example, with $n = 12$ and $|PRT_k| = 4$, the size of the set of optimally balanced solutions, $|F^{*}|$, when using the *a priori* data could be calculated as $(12 \cdot 9 \cdot 6 \cdot 3) \cdot (8 \cdot 6 \cdot 4 \cdot 2) \cdot (4 \cdot 3 \cdot 2 \cdot 1) = 17{,}915{,}904$ from table 1.

Table 1. No. of possible entries in each element position resulting in Perfect Balance using *a priori* data with $n = 12$ and $|PRT_k| = 4$

k	1	2	3	4	5	6	7	8	9	10	11	12
count	12	9	6	3	8	6	4	2	4	3	2	1

Grouping these counts by workstation and reversing their ordering enables one to more easily recognize a pattern.

(1 2 3 4)
(2 4 6 8)
(3 6 9 12)

It can be seen that the first row can be generalized as $(1 \cdot 2 \cdot 3 \cdot \ldots \cdot |PRT_k|)$, the second as $(2 \cdot 4 \cdot 6 \cdot \ldots \cdot 2 \cdot |PRT_k|)$, and the third as $(3 \cdot 6 \cdot 9 \cdot \ldots \cdot 3 \cdot |PRT_k|)$. Expanding in this way, the number of optimally balanced solutions can be written as:

$$|F^*| = (1 \cdot 2 \cdot 3 \cdot \ldots \cdot (1 \cdot |PRT_k|)) \cdot (2 \cdot 4 \cdot 6 \cdot \ldots \cdot (2 \cdot |PRT_k|)) \cdot \ldots \cdot \left(\frac{n}{|PRT_k|} \cdot \frac{2n}{|PRT_k|} \cdot \frac{3n}{|PRT_k|} \ldots \cdot n \right)$$

This can be written as:

$$|F^*| = \prod_{x=1}^{|PRT_k|} x \cdot \prod_{x=1}^{|PRT_k|} 2x \cdot \prod_{x=1}^{|PRT_k|} 3x \cdot \ldots \cdot \prod_{x=1}^{|PRT_k|} \frac{n}{|PRT_k|} \cdot x$$

and finally as:

$$|F^*| = \prod_{x=1}^{|PRT_k|} x^{\frac{n}{|PRT_k|}} \cdot \prod_{y=1}^{\frac{n}{|PRT_k|}} y$$

or:

$$|F^*| = \prod_{x=1}^{|PRT_k|} \prod_{y=1}^{\frac{n}{|PRT_k|}} x^{\frac{n}{|PRT_k|}} \cdot y \tag{18}$$

Since $|PRT_k| = 4$ in this paper, formula (18) becomes:

$$|F^*| = \prod_{x=1}^{4} \prod_{y=1}^{\frac{n}{4}} x^{\frac{n}{4}} \cdot y \tag{19}$$

In our example with $n = 12$ and $|PRT_k| = 4$, formula (19) is solved as:

$$|F^*| = \prod_{x=1}^{4} \prod_{y=1}^{\frac{12}{4}} x^{\frac{12}{4}} \cdot y = \prod_{x=1}^{4} \prod_{y=1}^{3} x^3 \cdot y = \prod_{x=1}^{4} x^3 \cdot (1 \cdot 2 \cdot 3)$$

or:

$$|F^*| = 1 \cdot 1 \cdot 1 \cdot (1 \cdot 2 \cdot 3) \cdot 2 \cdot 2 \cdot 2 \cdot (1 \cdot 2 \cdot 3) \cdot 3 \cdot 3 \cdot 3 \cdot (1 \cdot 2 \cdot 3) \cdot 4 \cdot 4 \cdot 4 \cdot (1 \cdot 2 \cdot 3)$$

which, when rearranged, can be written as the more familiar:

$$|F^*| = (12 \cdot 9 \cdot 6 \cdot 3) \cdot (8 \cdot 6 \cdot 4 \cdot 2) \cdot (4 \cdot 3 \cdot 2 \cdot 1) = 17{,}915{,}904$$

Although the size of the *a priori* optimal sequence set is quite large in this example, it is also significantly smaller than the search space of $n! = 479{,}001{,}600$. As shown in table 2, the number of sequences that are optimal in balance goes from 100% of n at $n = 4$, to 22.9% at $n = 8$, and to less than 1% at $n = 16$; as n grows, this percentage gets closer and closer to 0%.

Table 2. Comparison of possible sequences to optimal sequences for a given *n* using *a priori* data

n	*n*!	Number of optimal sequences	Percentage of optimal sequences
4	24	24	100.00%
8	40,320	9,216	22.86%
12	479,001,600	17,915,904	3.74%
16	2.09228E+13	1.10075E+11	0.53%

9.1. Known optimal benchmark data results

An analysis was performed using the previously described data set. Each of the combinatorial optimization methods was tested on the full range of data sets and compared to each other and to the known best and worst case performance on the basis of: total number of workstations, the McGovern-Gupta measure of balance, and time complexity. The methods with probabilistic components (ACO and GA) were run three times with the reported results being the mean of these runs. In addition, all techniques were run three times to provide mean time complexity data.

The first study performed was a measure of each of the techniques' calculated number of workstations as compared to the optimal as given by formula (12). As shown in Fig. 2, all of the methods performed very well in workstation calculation, staying within 2 workstations of optimum for all data set sizes tested.

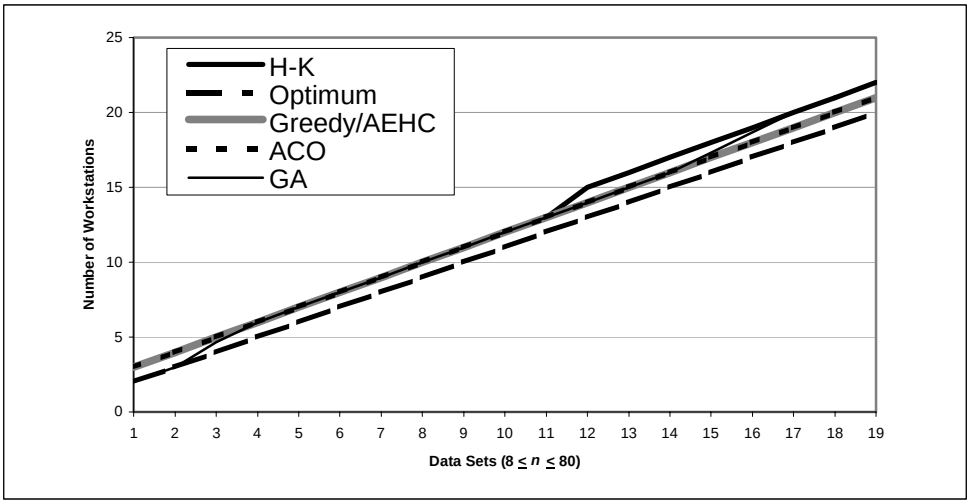


Fig. 2. Workstation calculations for each DLBP combinatorial optimization method.

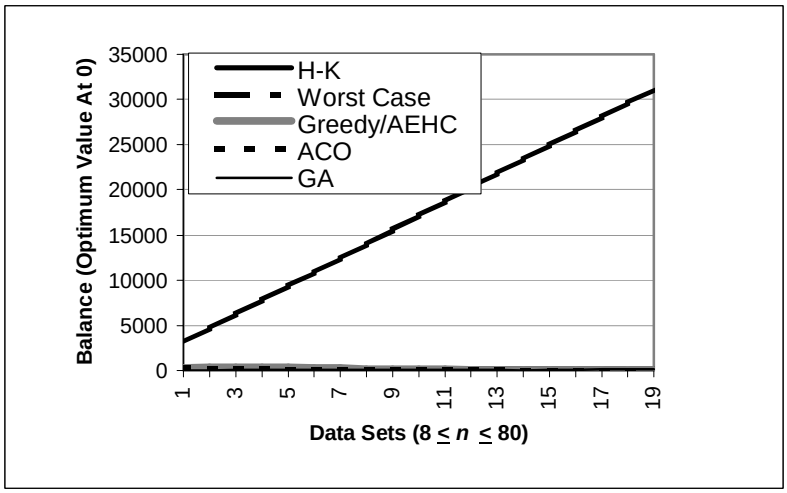


Fig. 3. DLBP combinatorial optimization methods' balance performance.

In terms of the calculated McGovern-Gupta measure of balance, again, all of the methods are seen to perform very well and significantly better than the worst-case (best-case is found uniformly at 0; see Fig. 3). However, examining the balance in greater detail provides some insight into the different techniques.

As seen in Fig. 4, ACO performed very well but (as seen later in the time complexity study) at the expense of time and extensive software modifications. Specifically, due to the nature of DLBP (i.e., with precedence constraints, a part's placement in a solution sequence cannot be evaluated until it is known what has come before it, and hence, how much time remains in the current workstation under consideration), formula (8) is made to be dynamic; that is, formula (8) for DLBP is calculated at each increment in time rather than just once at the start of each cycle. Although slightly more time consuming, this results in excellent solution results. Also, additional time is required by DLBP ACO since twice as many edges are evaluated than would be in a general-purpose ACO since all edges in DLBP are directed edges with *pq* not necessarily equivalent to *qp*.

It can also be seen that the performance of the DLBP versions of ACO and the Greedy/AEHC hybrid improves with problem size, while the DLBP H-K and GA tend to decrease similarly and in a stepwise fashion (note that their performance, in fact, improves overall with problem size as a percentage of worst case).

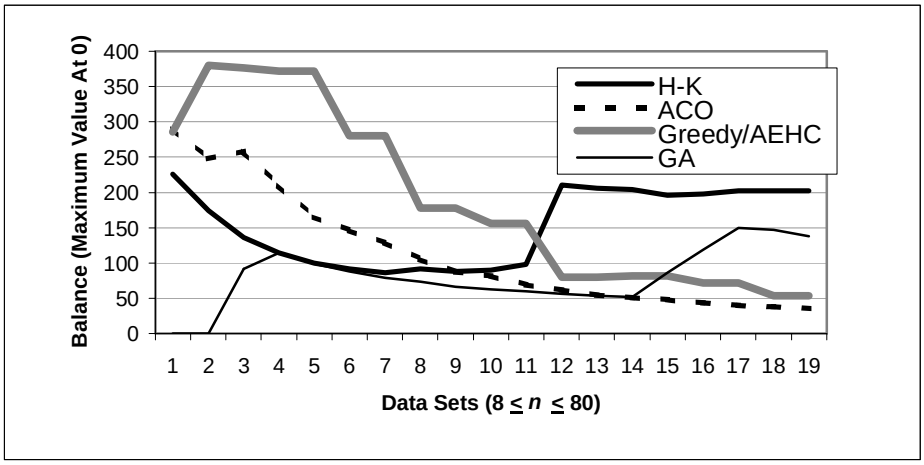


Fig. 4. Detailed DLBP combinatorial optimization methods' balance performance.

Finally, time complexity was examined using the *a priori* data. All of the techniques were seen to be very fast; Fig. 5. Each is significantly faster than exhaustive search and each is seen to be faster than $O(n^3)$.

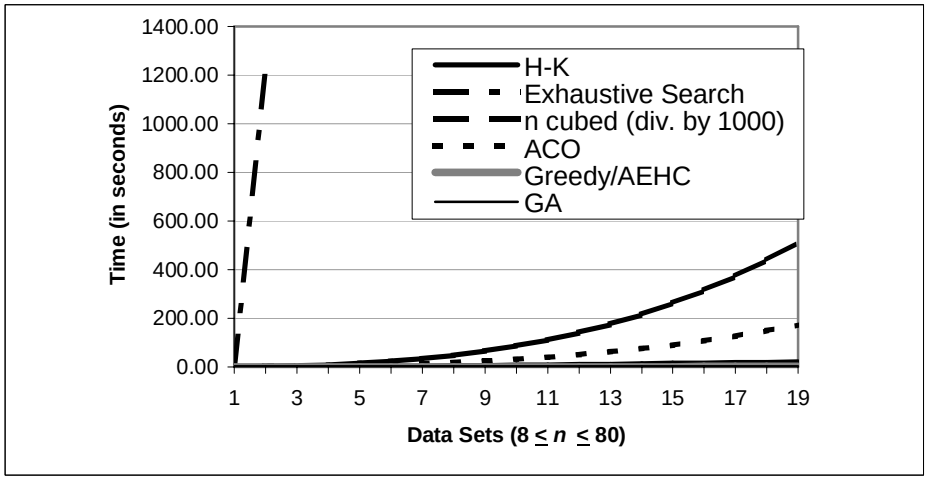


Fig. 5. Time complexity of DLBP combinatorial optimization methods.

The time complexity can be examined in greater detail in Fig. 6. A heuristic combination specifically designed for the DLBP, the Greedy/AEHC hybrid proved to be, as expected, the fastest technique examined, however, DLBP GA was

seen to be only slightly slower. DLBP H-K was also very fast, even with $\Delta\psi$ varying from 1 to 10 and with forward and reverse data runs (the H-K process grows approximately exponentially in $1/\psi$, taking, for example from $1/10^{th}$ of a second at ψ = 5 up to over 7 seconds at ψ = 2 with n = 12; exhaustive search was seen to take over 20 minutes on the same size data). The slowest was shown to be DLBP ACO, but this can be attributed to the dynamic probability calculations, which is more indicative of the challenges posed by DLBP than any concerns with the methodology.

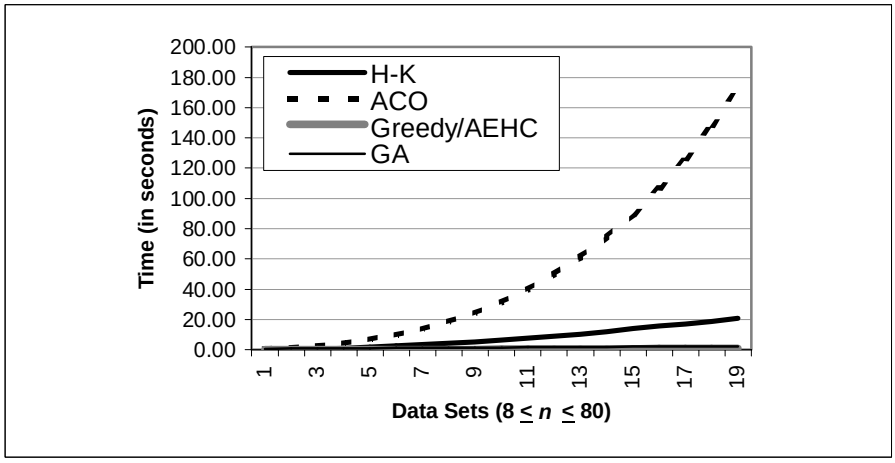


Fig. 6. Detailed time complexity of DLBP combinatorial optimization methods.

Although the preceding balance results are not optimal, this is more a reflection of the especially 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 is intended to pose challenges, even at relatively small n values, to a variety of solution-generating techniques. The inclusion of precedence constraints will increasingly move any of these methods towards the optimal solution. Larger n will increasingly move the DLBP Greedy algorithm/AEHC heuristic hybrid and DLBP ACO towards the optimal solution, while a smaller ψ or multiple runs with randomized (or other formatted) data sets will increasingly move the DLBP H-K metaheuristic towards the optimal solution.

Time complexity improvements can be gained: in DLBP GA with smaller generations; a smaller number of cycles or probability calculations that are not dynamic for DLBP ACO; increases in ψ for DLBP H-K. Each of these can be expected to be at the cost of solution performance. Due to its problem-specific design, there are no obvious ways to speed up the DLBP Greedy algorithm/AEHC heuristic hybrid, however, a gain can be easily realized through replacing its typically slow sort routine (Bubble Sort) to a faster algorithm such as Quick Sort. Additional DLBP GA and ACO analysis can be found in [[22]], while studies including DLBP H-K and Greedy/AEHC can be found in [[24]].

10. CONCLUSIONS

Four very fast, near-optimal, combinatorial optimization methods for solving the deterministic DLBP were developed, presented, and compared in this paper. These methods include ACO, GA, and H-K metaheuristics modified for the DLBP, as well as a problem-specific Greedy/AEHC hybrid. Each method seeks to provide a feasible part removal sequence while attempting to find the best McGovern-Gupta measure of balance. Although sub-optimum techniques, these combinatorial optimization methods quickly found near-optimal solutions in a variety of exponentially large search spaces. These methods appear well suited to the decision making problem format as well as for the solution of problems with nonlinear objectives. In addition, they are 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

[1] C. Bierwirth, D. C. Mattfeld, and H. Kopfer, On Permutation Representations For Scheduling Problems, Parallel Problem Solving from Nature, in H. M. Voigt, W. Ebeling, I. Rechenberg, and H. P. Schwefel, eds., *Lecture Notes in Computer Science*, Berlin, Germany, Springer-Verlag, Vol 1141, pp. 3.10-3.18, 1996.

[2] L. Brennan, S. M. Gupta, and K. N. Taleb, "Operations Planning Issues In An Assembly/Disassembly Environment," *International Journal of Operations and Production Planning*, Vol 14, No. 9, pp. 57-67, 1994.
[3] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms*, The MIT Press, Cambridge, MA, 2001.
[4] M. Dorigo and G. Di Caro, The Ant Colony Optimization Meta-Heuristic, in D. Corne, M. Dorigo, and F. Glover, eds., *New Ideas in Optimization*, McGraw-Hill, Maidenhead, UK, pp. 11-32, 1999.
[5] M. Dorigo, V. Maniezzo, and A. Colorni, "The Ant System: Optimization By A Colony Of Cooperating Agents," *IEEE Transactions on Systems, Man, and Cybernetics–Part B*, Vol 26, No. 1, pp. 1-13, 1996.
[6] E. A. Elsayed and T. O. Boucher, *Analysis and Control of Production Systems*, Prentice Hall, Upper Saddle River, NJ, 1994.
[7] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, New York, NY, 1979.
[8] A. Gungor and S. M. Gupta, "Issues In Environmentally Conscious Manufacturing And Product Recovery: A Survey," *Computers and Industrial Engineering*, Vol 36, No. 4, pp. 811-853, 1999.
[9] A. Gungor and S. M. Gupta, "Disassembly Line Balancing," *Proceedings of the 1999 Annual Meeting of the Northeast Decision Sciences Institute*, March 24-26, Newport, Rhode Island, pp. 193-195, 1999.
[10] A. Gungor and S. M. Gupta, "A Systematic Solution Approach to the Disassembly Line Balancing Problem," *Proceedings of the 25th International Conference on Computers and Industrial Engineering*, March 29-April 1, New Orleans, Louisiana, pp. 70-73, 1999.
[11] A. Gungor and S. M. Gupta, "A Solution Approach To The Disassembly Line Problem In The Presence Of Task Failures," *International Journal of Production Research*, Vol 39, No. 7, pp. 1427-1467, 2001.
[12] A. Gungor, and S. M. Gupta, "Disassembly Line In Product Recovery," *International Journal of Production Research*, Vol 40, No. 11, pp. 2569-2589, 2002.
[13] S. M. Gupta and S. M. McGovern, "Multi-Objective Optimization In Disassembly Sequencing Problems," *Proceedings of the 2nd World Conference on Production & Operations Management and the 15th Annual Production & Operations Management Conference*, April 30-May 3, Cancun, Mexico, CD-ROM, 2004.
[14] S. M. Gupta and K. N. Taleb, "Scheduling Disassembly," *International Journal of Production Research*, Vol 32, pp. 1857-1866, 1994.
[15] A. A. Hopgood, *Knowledge-Based Systems For Engineers And Scientists*, CRC Press, Boca Raton, FL, 1993.
[16] E. Kongar and S. M. Gupta, "A Genetic Algorithm For Disassembly Process Planning," *Proceedings of the 2001 SPIE International Conference on Environmentally Conscious Manufacturing II*, October 28-29, Newton, Massachusetts, pp. 54-62, 2001.
[17] J. R. Koza, *Genetic Programming: On The Programming Of Computers By The Means Of Natural Selection*, MIT Press, Cambridge, MA, 1992.
[18] S. M. McGovern and S. M. Gupta, "2-Opt Heuristic for the Disassembly Line Balancing Problem," *Proceedings of the SPIE International Conference on Environmentally Conscious Manufacturing III*, October 27-31, Providence, Rhode Island, pp. 71-84, 2003.
[19] S. M. McGovern and S.M. Gupta, "Demanufacturing Strategy Based Upon Metaheuristics," *Proceedings of the 2004 Industrial Engineering Research Conference*, May 15-19, Houston, Texas, CD-ROM, 2004.
[20] S. M. McGovern and S. M. Gupta, "Greedy Algorithm for Disassembly Line Scheduling," *Proceedings of the 2003 IEEE International Conference on Systems, Man, and Cybernetics*, October 5-8, Washington, DC, pp. 1737-1744, 2003.
[21] S. M. McGovern and S. M. Gupta, "Metaheuristic Technique For The Disassembly Line Balancing Problem," *Proceedings of the 2004 Northeast Decision Sciences Institute Conference*, March 24-26, Atlantic City, New Jersey, pp. 223-225, 2004.
[22] S. M. McGovern and S. M. Gupta, "Multi-Criteria Ant System And Genetic Algorithm For End-Of-Life Decision Making," *Proceedings of the 35th Annual Meeting of the Decision Sciences Institute*, November 20-23, Boston, Massachusetts, 2004.
[23] S. M. McGovern, S. M. Gupta, and S. V. Kamarthi, "Solving Disassembly Sequence Planning Problems Using Combinatorial Optimization," *Proceedings of the 2003 Annual Meeting of the Northeast Decision Sciences Institute*, March 27-29, Providence, Rhode Island, pp. 178-180, 2003.
[24] S. M. McGovern, S. M. Gupta, and K. Nakashima, "Multi-Criteria Optimization for Non-Linear End of Lifecycle Models," *Proceedings of the Sixth Conference on EcoBalance*, October 25-27, Tsukuba, Japan, 2004.