

January 01, 2004

Multi-criteria ant system and genetic algorithm for end-of-life decision making

Surendra M. Gupta
Northeastern University

Seamus M. McGovern
Northeastern University

Recommended Citation

Gupta, Surendra M. and McGovern, Seamus M., "Multi-criteria ant system and genetic algorithm for end-of-life decision making" (2004).. Paper 74. <http://hdl.handle.net/2047/d10013672>

This work is available open access, hosted by Northeastern University.



Laboratory for Responsible Manufacturing

Bibliographic Information

McGovern, S. M. and Gupta, S. M., "Multi-Criteria Ant System and Genetic Algorithm for End-of-Life Decision Making", ***Proceedings of the 2004 Decision Sciences Institute Conference***, Boston, Massachusetts, pp. 6371-6376, November 20-23, 2004.

Copyright Information

Copyright 2004, Surendra M. Gupta.

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**

MULTI-CRITERIA ANT SYSTEM AND GENETIC ALGORITHM FOR END-OF-LIFE DECISION MAKING

Seamus M. McGovern, Northeastern University, Boston, MA 02115, mcgovern.s@neu.edu, (617)-494-2054
Surendra M. Gupta*, Northeastern University, Boston, MA 02115, gupta@neu.edu, (617)-373-4846

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: is feasible, minimizes workstations, and ensures similar idle times, as well as other, disassembly-specific concerns. Finding the optimal balance is computationally intensive due to factorial growth. Ant colony optimization and genetic algorithm metaheuristics are presented and compared along with an example to illustrate the implementation. Conclusions drawn include the consistent generation of optimal or near-optimal solutions, the ability to preserve precedence, the speed of the metaheuristics, and their practicality due to ease of implementation.

Keywords: Disassembly, disassembly line balancing, ant colony optimization, genetic algorithm

1. INTRODUCTION

Disassembly is the methodical extraction of valuable parts and materials from discarded products. In this paper, the disassembly line balancing problem (DLBP) is solved using ant colony optimization (ACO) and genetic algorithm (GA) metaheuristics. While exhaustive search consistently provides the problem's optimal solution, its time complexity quickly limits its practicality. The application of these metaheuristics 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 [3]. The GA involves a randomly generated initial population with crossover and mutation performed over many generations. Both seek to preserve precedence relationships and are further modified to address multiple objectives.

2. LITERATURE REVIEW

Brennan *et al.* [2] and Gupta and Taleb [13] investigated the problems associated with disassembly planning and scheduling. Gungor and Gupta [8], [9], [11] presented the first introduction to the DLBP and then developed an algorithm for solving it in the presence of failures [10]. For a review of environmentally conscious manufacturing and product recovery see Gungor and Gupta [7]. McGovern *et al.* first applied combinatorial optimization techniques to the DLBP [18]. Gutjahr and Nemhauser [14] described a solution to the assembly line balancing problem using a technique similar to dynamic programming. Erel and Gokcen [6] developed a modified version of Gutjahr and Nemhauser's line balancing algorithm by allowing for mixed-model lines. Suresh *et al.* [19] presented a GA to provide a near-optimal solution to the assembly line balancing problem models.

3. NOTATION

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

\mathbf{a}	weight of existing pheromone (trail) in path selection
\mathbf{b}	weight of a given edge in path selection
\mathbf{h}_{pq}	ACO visibility value of edge pq
\mathbf{r}	variable such that $1 - \mathbf{r}$ represents the pheromone evaporation rate
$\mathbf{t}_{pq}(NC)$	amount of trail on edge pq during cycle NC
c	initial amount of pheromone on each of the paths
CT	cycle time; maximum time available at each workstation
d_k	demand; quantity of part k requested
D	demand rating for a given solution sequence
F	McGovern-Gupta measure of balance for a given solution sequence
F_{tr}	McGovern-Gupta measure of balance of ant r sequence at time t
h_k	binary value; 1 if part k is hazardous, else 0
H	hazard rating for a given solution sequence
j	workstation count ($1, \dots, NWS$)

* Corresponding author

k	part identification ($1, \dots, n$)
L_r	ACO delta trail divisor value; here set equal to F_{nr}
m	number of ants
n	number of parts for removal
N	number of chromosomes (population size)
\mathbb{N}	the set of natural numbers
NC_{max}	maximum number of cycles
NWS	number of workstations required for a given solution sequence
$P_{pq}^r(t)$	probability of ant r taking an edge pq at time t
PRT_k	part removal time required for k th part
PS_k	k^{th} part in a solution sequence (i.e., for solution “3, 1, 2” $PS_2 = 1$)
Q	amount of pheromone added if a path is selected
r	ant count; also, integer reference for r -Opt (e.g., 2-Opt)
R_m	mutation rate
R_x	crossover rate
t	time within a cycle; ranges from 0 to n
WS_j	elapsed time in workstation j

4. THE DLBP MODEL DESCRIPTION

The particular model investigated in this paper seeks to fulfill four objectives:

1. minimize the number of disassembly workstations and hence, minimize the total idle time,
2. ensure the idle times at each workstation are similar,
3. remove hazardous components early in the disassembly sequence, and
4. remove high demand components before low demand components,

with a major constraint being to provide a feasible sequence. The result is an integer, multiple criteria decision-making problem. 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 [5]. SI rewards similar idle times at each workstation but at the expense of allowing for a large number of workstations. This is because SI compares workstation elapsed times to the largest WS_j instead of to CT . IE rewards the minimum number of workstations, but allows unlimited variance in idle times between workstations because no comparison is made between WS_j s. This paper makes use of a balancing method [17] that simultaneously minimizes the number of workstations while ensuring that idle times at each workstation are similar, though at the expense of the generation of a nonlinear objective function. The method 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. Therefore, a resulting minimum value is the more desirable solution indicating both a minimum NWS and similar idle times. The McGovern-Gupta measure of balance is:

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

The hazard measure is represented as:

$$H = \sum_{k=1}^n (k \cdot h_{PS_k}) \quad h_{PS_k} = \begin{cases} 1, & \text{hazardous} \\ 0, & \text{otherwise} \end{cases} \quad (2)$$

The demand measure is represented as:

$$D = \sum_{k=1}^n (k \cdot d_{PS_k}) \quad d_{PS_k} \in \mathbb{N}, \forall PS_k \quad (3)$$

The overall goal is to minimize each of these measures, with a priority of F , then H , then D . Gupta and McGovern [12] provide further explanation of formulae (1) through (3).

5. METAHEURISTICS SELECTION AND COMPARISON

Metaheuristics allow the optimal or near-optimal solution of many NP-Hard problems including many decision-making problems. ACO and GA were selected for comparison due to a variety of similarities as well as some distinct differences. ACO and GA (unlike, for example, Tabu search) generate the best solutions when sub-optimal solutions bear a resemblance to the optimal solution. This is seen in ACO by the ants working to reinforce good solution sections by adding trail (similar to pheromones in actual ants). It is performed in GA through the breaking

apart of good solutions to be recombined with other good solution sections. The drawback to these is that if good sub-optimal solutions do not bear a resemblance to the optimal solution, the optimal solution may not be found, though in many applications it is not typical for the data to perform in this manner. Also, isolated solutions are addressed using probabilistically selected tours, allowing for potential excursions to seemingly poor performing areas of the search space. Unlike simulated annealing (SA) but similarly to Tabu, ACO and GA don't look at a large area initially then smaller and smaller areas in more detail. Unlike Tabu search, ACO and GA can revisit solutions found previously. Unlike GA, many applications of ACO (as well as Tabu and SA) generate solutions similar to branch-and-bound where the best solution is grown by making decisions on which branch to take as the sequence moves from beginning to end. Rather, in GA an entire solution sequence is generated prior to evaluation. Similarly, ACO employs a greedy component since movement through the search space requires branching decisions that are typically made based upon the best short-term (greedy) move. Unlike greedy and r -Opt heuristics and traditional mathematical programming techniques such as linear programming, ACO and GA have a strong probabilistic component. As a result, they do not proceed in a deterministic manner and do not necessarily provide the same solution every time they are run. Unlike traditional mathematical programming techniques, ACO and GA are not limited to linear problem descriptions. Like most metaheuristics, they cannot consistently provide the optimal solution, as is commonly the case with most mathematical programming. Also common to metaheuristics, there is some degree of tuning and design required; for example, selection of number of generations, mutation rate, crossover rate, and crossover method in GA; or pheromone evaporation rate, number of ants, number of cycles, and visibility definition for ACO.

6. THE DLBP ACO ALGORITHM

6.1. 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 a 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 (4). Based on these, the next step 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 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:

$$p_{pq}^r(t) = \begin{cases} \frac{[t_{pq}(t)]^a \cdot [h_{pq}]^b}{\sum_{r \in allowed_r} [t_{pr}(t)]^a \cdot [h_{pr}]^b} & q \in allowed_r \\ 0 & otherwise \end{cases} \quad (4)$$

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

$$t_{pq}(NC+1) = r \cdot t_{pq}(NC) + \Delta t_{pq} \quad (5)$$

where:

$$\Delta t_{pq} = \sum_{r=1}^m \Delta t_{pq}^r \quad \text{and:} \quad \Delta t_{pq}^r = \begin{cases} \frac{Q}{L_r} & \text{edge}(p,q) \text{ used} \\ 0 & \text{otherwise} \end{cases} \quad (6)$$

6.2. DLBP-specific modifications

DLBP ACO is modified to account for feasibility constraints and provide for multiple objectives. In DLBP, a solution consists of a sequence of work elements (also: 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 the 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 until NC_{max} . For DLBP ACO, the visibility, h_{pq} , was defined as:

$$h_{pq} = \frac{1}{F_{tr}} \quad (7)$$

while in formula (6), L_r was 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 h_{pq} are related here, this is not unusual in ACO applications (in the traveling salesman problem, L_r is the tour length while h_{pq} is the reciprocal of a the distance between cities p and q). However, this method of selecting h_{pq} , effectively a short-term greedy choice, may not always translate into the best final solution for a complex problem like DLBP. NC_{max} was set to 300 since larger problems than that studied here were shown to reach their best solution by that count. The process was not run until no improvements were shown but, as is the norm with many combinatorial optimization techniques, was 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.* [4], the following was used: $a = 1.00$, $b = 5.00$, $r = 0.50$, $Q = 100.00$, and $c = 1.00$.

7. THE DLBP GENETIC ALGORITHM

7.1. GA model description

A GA has a solution structure defined as a chromosome, which is made up of genes (parts) and generated by two parent chromosomes (each with its own measure of fitness based on the section 4 objectives) from the pool (population) of solutions. New solutions are made from old using crossover (sever parents genes and swap severed sections) and mutation (randomly vary genes within a chromosome). Only feasible disassembly sequences were considered as members of the population. The fitness was computed for each chromosome using the same evaluation criteria as in DLBP ACO. An initial, feasible population was randomly generated and the fitness of each chromosome in this generation was then calculated. An even integer of R_xN parents was randomly selected for crossover to produce R_xN offspring (offspring make up $R_xN \times 100\%$ of each generations' 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 (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 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 (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.

7.2. DLBP-specific modifications

DLBP GA was modified from a general 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, the 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% [16], 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.

8. NUMERICAL RESULTS

8.1. Case study and DLBP ACO and DLBP GA results

The developed metaheuristics were investigated on a variety of test cases to confirm performance and optimize parameters. They were then used to provide a solution to the DLBP where the objective is to completely disassemble a product consisting of $n = 8$ subassemblies on a disassembly line operating at a speed which allows $CT = 40$ seconds for each workstation. This provided an application to an actual disassembly line balancing problem. This practical and relevant example is based on the disassembly of a personal computer (PC) consisting of subassemblies with part removal times of $PRT_k = \{14, 10, 12, 18, 23, 16, 20, 36\}$, binary hazard values of $h_k = \{0, 0, 0, 0, 0, 0, 1, 0\}$, and part demands of $d_k = \{360, 500, 620, 480, 540, 750, 295, 720\}$. See [11] for precedence constraints and a more detailed description.

Over multiple runs, both DLBP ACO and DLBP GA obtained all four solutions optimal in F (table 1) and did so very quickly. The speed for the C++ implemented program on this problem was less than $1/10^{\text{th}}$ of a second (averaging 0.04 seconds per run) for DLBP ACO on a 1.6GHz PM x86 family workstation, and (searching 1000 generations of 20 chromosomes each) just over one second for DLBP GA. For a more in-depth study of these metaheuristics and analysis on larger and more diverse problem sets, see [18] and [12].

Table 1. The four disassembly sequence solutions optimal in F

a)		Workstation				Time to remove part (in seconds)
		1	2	3	4	
Part removal sequence	1	14				
	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	

b)		Workstation				Time to remove part (in seconds)
		1	2	3	4	
Part removal sequence	1	14				
	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	

c)		Workstation				Time to remove part (in seconds)
		1	2	3	4	
Part removal sequence	1	14				
	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	

d)		Workstation				Time to remove part (in seconds)
		1	2	3	4	
Part removal sequence	1	14				
	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	

Due to its probabilistic component, DLBP ACO was seen to select a solution sub-optimal in F about once in every twenty runs. It also did not find the optimal solutions with the same frequency, finding the solution in table 1a most often, followed by 1b, then 1c, then 1d. This can be attributed to formula (7) and the ACOs requirement to evaluate

partial solutions before making a solution element selection decision, as well as the higher demand performance of the solution in table 1a. In addition, it was also observed that edges not selected are diluted very rapidly; it was not unusual to see the bulk of the edges (all initialized to $c = 1$) with trail values of 1×10^{-91} after the 300 cycles. The developed GA also obtained all four solutions optimal in F over multiple runs. With 20 chromosomes, this resulted in a total of 8 best answers carried over from generation to generation with the 60% crossover rate using $R_x N$ (i.e., 0.6×20 , or 12) parents and therefore produced 12 offspring. The GA converged to moderately good solutions very quickly. It was able to find at least one of the four F optimal solutions after no more than 10 generations. 100 generations consistently provided three to four of the solutions, while 1000 generations almost always resulted in the generation of all four.

REFERENCES

- [1] Bierwirth, C., Mattfeld, D.C. & Kopfer, H. *On Permutation Representations For Scheduling Problems*. Parallel Problem Solving from Nature. Voigt, H.M., Ebeling, W., Rechenberg, I. & Schwefel, H.P. eds. Lecture Notes in Computer Science. Berlin, Germany: Springer-Verlag, 1996, 1141, 3.10-3.18.
- [2] Brennan, L., Gupta, S.M. & Taleb, K.N. *Operations Planning Issues In An Assembly/Disassembly Environment*. International Journal of Operations and Production Planning, 1994, 14(9), 57-67.
- [3] Dorigo, M. & Di Caro, G. The Ant Colony Optimization Meta-Heuristic, In Corne, D., Dorigo, M. & Glover, F. eds. New Ideas in Optimization. Maidenhead, UK: McGraw-Hill, 1999, 11-32.
- [4] Dorigo, M., Maniezzo, V. & Coloni, A. *The Ant System: Optimization By A Colony Of Cooperating Agents*. IEEE Transactions on Systems, Man, and Cybernetics—Part B, 1996, 26(1), 1-13.
- [5] Elsayed, E.A. & Boucher, T.O. *Analysis and Control of Production Systems*. Upper Saddle River, NJ: Prentice Hall, 1994.
- [6] Erel, E. & Gokcen, H. *Shortest-Route Formulation Of Mixed-Model Assembly Line Balancing Problem*. Management Science, 1964, 11(2), 308-315.
- [7] Gungor, A. & Gupta, S.M. *Issues In Environmentally Conscious Manufacturing And Product Recovery: A Survey*. Computers and Industrial Engineering, 1999, 36(4), 811-853.
- [8] Gungor, A. & Gupta, S.M. *Disassembly Line Balancing*. Proceedings of the 1999 Annual Meeting of the Northeast Decision Sciences Institute, Newport, Rhode Island, March 24-26, 1999, 193-195.
- [9] Gungor, A. & Gupta, S.M. *A Systematic Solution Approach To The Disassembly Line Balancing Problem*. Proceedings of the 25th International Conference on Computers and Industrial Engineering, New Orleans, Louisiana, March 29-April 1, 1999, 70-73.
- [10] Gungor, A. & Gupta, S.M. *A Solution Approach To The Disassembly Line Problem In The Presence Of Task Failures*. International Journal of Production Research, 2001, 39(7), 1427-1467.
- [11] Gungor, A. & Gupta, S.M. *Disassembly Line In Product Recovery*. International Journal of Production Research, 2002, 40(11), 2569-2589.
- [12] Gupta, S.M. & McGovern, S.M. *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, Cancun, Mexico, April 30 – May 3, 2004, CD-ROM.
- [13] Gupta, S.M. & Taleb, K.N. *Scheduling Disassembly*. International Journal of Production Research, 1994, 32, 1857-1866.
- [14] Gutjahr, A.L. & Nemhauser, G.L. *An Algorithm For The Line Balancing Problem*. Management Science, 1964, 11(2), 308-315.
- [15] Hopgood, A.A. *Knowledge-Based Systems For Engineers And Scientists*. Boca Raton, FL: CRC Press, 1993.
- [16] Koza, J.R. *Genetic Programming: On The Programming Of Computers By The Means Of Natural Selection*. Cambridge, MA: MIT Press, 1992.
- [17] McGovern, S.M. & Gupta, S.M. *Greedy Algorithm For Disassembly Line Scheduling*. Proceedings of the 2003 IEEE International Conference on Systems, Man, and Cybernetics, October 5-8, Washington, DC, 2003, 1737-1744.
- [18] McGovern, S.M., Gupta, S.M. & Kamarathi, S.V. *Solving Disassembly Sequence Planning Problems Using Combinatorial Optimization*. Proceedings of the 2003 Northeast Decision Sciences Institute Conference, March 27-29, Providence, Rhode Island, 2003, 178-180.
- [19] Suresh, G., Vinod, V.V. & Sahu, S. *A Genetic Algorithm For Assembly Line Balancing*. Production Planning and Control, 1996, 7(1), 38-46.