12th IFAC Symposium on Information Control Problems in Manufacturing – INCOM 2006, Saint-Etienne, France, May 17 – 19, 2006.

SELECTION OF A MODULES STOCK COMPOSITION USING GENETIC ALGORITHM

Bruno Agard[1], Bernard Cheung[1,2], Catherine da Cunha[1,3]

[1] Polygistique, [2] GERAD, École Polytechnique de Montréal,
C.P. 6079, succ. Centre-ville, Montréal (Québec), H3C 3A7, Canada
[3] Laboratoire GILCO-INPG, 46 av Félix Viallet, 38031 Grenoble Cedex 1, France

Abstract: The paper focuses on modelling and solving a design problem. The problem consists in selecting a set of modules that will be manufactured in distant sites and shipped in a nearby location site for a final assembly operation under time limits. The problem is modelled as a mathematical problem and solved by a genetic algorithm with a modified crossover operation, a uniform mutation with adaptive rate and a partial reshuffling procedure. *Copyright © 2006 IFAC*

Keywords: Modular Design, Genetic Algorithm, Modified Crossover Operator, Uniform Mutation with Adaptive Rate, Partial Reshuffling Procedure.

1. INTRODUCTION

Actual competition scheme is such that a company has to provide the exact product at the exact place and at the exact moment (Pine, 1993).
Moreover the company may have to take into account different production sites with different production capacities and production costs that may be located anywhere in the world, with more or less time to response.
Consider a company that has to provide a large product portfolio in order to get closer to each potential customer. That company is structured with different production sites located in different areas around the world. The production policy of that company is such that the distant location sites manufacture pre-assembly components (modules) that are shipped to a nearby location site in order to be assembled in a final operation according to the desires of each customer (See Fig. 1) with a fixed maximum time of deliverance for any product.

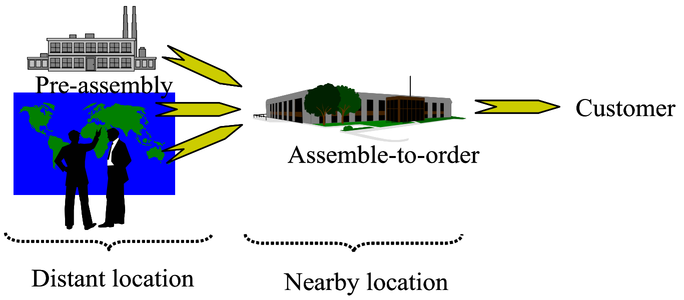

Fig. 1. Industrial configuration.

Dealing with a large product portfolio and a short time of deliverance is an area of mass customization (Pine, 1993).
Modular design and product delayed differentiation may be combined in assemble to order. Assemble to order makes it possible to provide a large number of final products from a limited number of modules (Starr, 1965). Each module may be manufactured in a different distant location site and shipped to get assembled according to customers order when an order is received.

To satisfy an order within respect of the delay may be a major issue (Cachon, 2003). The structure of the supply chain (licked to the number of modules and to their constitution) impacts the delivery time (through the final assembly time). This article exploits the performance of genetic algorithms to design a supply chain from the selection of a module composition that minimizes the mean final assembly time.

2. PROBLEM MODELLING

The problem describe above in Figure 1 is modeled as follows (See Fig 2. (da Cunha and Agard, 2005)).

Consider the following notations: a_i is a component, $i \in [0; n]$, n is the number of components. The components may be delivered to any distant location site or directly to the nearby location site.
At the distant location site, the components are combined together to provide *SV* modules.

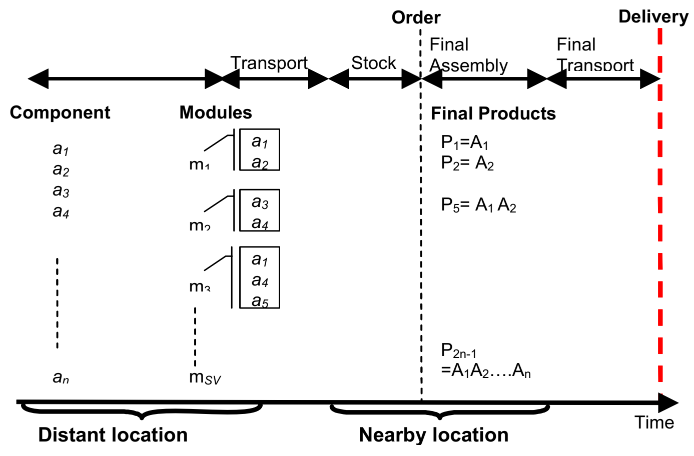

Fig. 2. Problem modelling

If there are no exclusive or inclusive relationships between components it is possible to have 2^n-1 different (non empty) products P_i from n components. P is the set of final products.
Call $p(P_i)$ the probability that a demand is for product $P_i \in$ P. The final demand can then be summarized by the scalar $D = (p(P_1)p(P_2) \ldots p(P_{2n-1}))$. The probability for each product comes from analysis of historical sales.
Let C be a module composition. Note $|C|$ the number of modules in stock.
Note $TA_D(C)$ the mean assembly time to assemble a demand D from the stock composition C.
A supplementary constraint is that no product could have any extra components or any doublet (only different options are assembled).

A stock composition C is evaluated with 2 criteria:

1. The number of modules, $|C|$
2. The mean assembly time, $TA_D(C)$.

For a fixed number of modules $|C|$ equal to SV (due to the nearby location site capacities), optimization concerns the minimization of the mean assembly time $TA_D(C)$.

The problem becomes, for a demand D, to select the module composition of size SV that minimizes the mean assembly time. For simplification, the assembly time is considered to be proportional to the number of assembly operations. Mean assembly time is then assimilated to the expected value of the number of assembly operations.

Note $NA(P_i;C)$ the number of assembly operations necessary to the realization of the product P_i starting from a module composition C.
Then $TA_D(C) = \sum_i p(P_i) \times NA(P_i, C)$
To get $NA(P_i;C)$ a Set partition problem as to be solved that gives the number of assembly operations needed to obtain P_i from C. This problem is NP-harp (Garey and Johnson, 1979). A greedy algorithm was chosen to evaluate $NA(P_i;C)$ (Johnson, 1974), (Chvatal, 1979). This approximation enables a rapid a solution.

3. PROBLEM SOLVING.

3.1 Encoding with a Genetic Algorithm.

In this section, we describe a modelling of the problem that enables to efficiently encode a genetic algorithm. The objective of the algorithm is to find for a given demand D and a given number of modules kept in inventory SV the optimal modules composition i.e. the composition that minimizes the mean assembly time.
A chromosome is a module composition C. It is represented by a binary vector of size $2^n - 1$, each digit stating the presence or absence of the given module in the composition. Let C be a module composition. $C[k] = 1$ when product P_k belongs to the composition, $C[k] = 0$ otherwise. The number of modules in stock can then be expressed as follows:

$$\left(|C| = \sum_{k=1}^{2^n-1} C[k]\right).$$

The fitness value of a given composition is defined as follows:

Fitnessvalue (C) =

$M - TA_D(C)$ *If the composition enables to assemble all the product portfolio*
0 *else*
where *M is a constant of big value*

As the goal is to minimize the mean assembly time, a greater fitness value means a better performance of the chromosome.
A chromosome that represents a stock composition with a size higher than SV receives a penalty ($M = 0$).

The crossover generation chosen here is a 2-point crossover, the points being randomly chosen (with a uniform law).
Let illustrate this operation with an example. Call C1 and C2 the parents, C3 and C4 the children.

C3[k]=C1[k], $1<k<i$ or $j<k<2^n$-1
C3[k]=C2[k], $i \le k \le j$

C4[k]=C2[k], $1<k<i$ or $j<k<2^n$-1
C4[k]=C1[k], $i \le k \le j$

3.2 Enhancement of the GA Search Scheme.

We recommend use of an enhancement genetic search scheme introduced by Cheung (2005) with appropriate crossover and mutation operators **for improved performance**. The details of these modifications are given in the following:

Modified Crossover Operator. Since all the genes in each chromosome assume **only 0 or 1 value**, it can be seen that a large proportion of chromosomes in a randomly generated population will have approximately half of its genes having same genes value of those at corresponding position in the optimal chromosome. **The crossover operation that exchanges a randomly selected fraction of genes between a pair of chromosomes at their corresponding position will be most beneficial**. By confining the crossover operations to a pair of distinct chromosomes (e.g. if the overlapping coefficient is lower than or equal to a prescribed value OR as described in previous section), then a) the redundancy in crossover operation will be almost eliminated, and b) the probability to obtain new solutions close to optimal are greatly enhanced. To visualize the effect of crossover action on a pair of binary chromosomes **a** and **b**, let Z_1 be the set of genes in **a** having common genes values with those of the optimal chromosome at the respective corresponding positions and let Z_2 be the set of genes in **b** having common genes values with those of the optimal chromosome at the respective corresponding positions. Let $Y_1 = Z_1 \setminus (Z_1 \cap Z_2)$ and $Y_2 = \setminus (Z_1 \cap Z_2)$, one can confine his consideration to the following standard case as illustrated below. Since for fixed i, j and overlap m, this case is unique up to a permutation of their relative positions. Without lost in generality, we may further assume that all the genes of the optimal chromosome have their value equal to one.

$$OC(C1,C2) = \frac{(2^n - 1) - (\sum_{k=1}^{2^n - 1} C1[k] - C2[k])}{2^n - 1}$$

Two identical chromosomes will then have an overlapping coefficient of 1, while chromosomes with no genes in common will have an overlapping coefficient of 0. The crossover between 2 chromosomes C1 and C2 will only be authorized if *OC(C1,C2) < OR*, *OR* being the overlapping rate, set for each implementation.

Uniform Mutation with Adaptive Rate. The uniform mutation changes the value of every gene positioned along the length of this chromosome with a probability μ. Thus, the uniform mutation changes randomly a fraction μ of the total number of gene values in a chromosome. That is, if N is the total number of genes, then randomly selected μN genes will be most likely to be mutated. The mutation operation tends to be disruptive as well as being constructive, appropriate rate of uniform mutation will allow those chromosomes to be mutated to have a higher probability of improving their fitness value. When the search is close to the optimal point, most of the chromosomes in the population will be very similar to the optimal chromosome, only a few bits of changes are necessary to transform them to the optimal one. One should adaptively reduce the mutation rate when this scenario occurs.

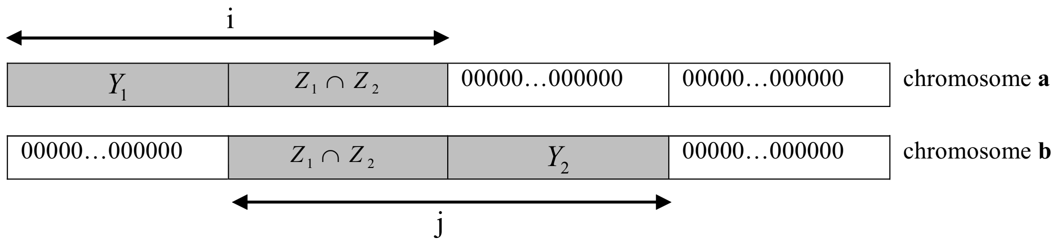

Fig. 3. Modified crossover operator.

Observe that if the overlap $Z1 \cap Z2$ is very small and if both the size of Y1 and Y2 is close to half of that of the entire chromosome, than any crossover action that swaps most of the elements of Y_2 to chromosome **a** (or Y_1 to chromosome **b**) will produce an offspring having most of the attributes of the optimal chromosome. (See Cheung, 2005 for detail).

To improve the performance of the 2-points crossover so that the crossover operations are limited to chromosomes' pairs that are not "too similar", an overlapping measure is defined. It evaluates the redundancy of 2 chromosomes that are to be paired. The overlapping coefficient of 2 chromosomes *C1* and *C2* is then defined as follows:

Partial Reshuffling Procedure. Our procedure takes advantage of the fact that considerable attributes of the optimal chromosome might have been acquired previously by some strings throughout the process of reproduction. These chromosomes having good solution potential should be allowed to recombine with some newly generated chromosomes for further improvements. Clearly, there is a considerable time saving when compared with the usual reshuffling procedure which requires an actual restart over from the very beginning. The detail of this procedure is described as follows.

For a given population of size 3n, the following steps are performed.

1. Generate n new strings $\{a_1, a_2, \ldots\ldots\ldots , a_n\}$ which are different from all the chromosomes in the original population, and arranged them in descending order of fitness.

2. For j = 1, 2, 3, …., n, if the fitness value of a_j is better than the worst string in the original population, then replace the worst string with this a_j . Otherwise, crossover a_j with all the string in the original population and then with all other a_i that are not equal to j. Update the population by replacing the worst string with the best string obtained each time.
3. If no more than 50% of the total replacement has been made possible, then generate more new chromosomes and repeat Step 2 until some more (say about 80%) of the less fitted original chromosomes are replaced.

Assuming that the generation process is random, and that the probability of obtaining improved new offspring when a mature chromosome crossovers with a new chromosome should be approximately the same as that when two new chromosome crossover with each other, it can be seen that the total number of crossovers in a partial reshuffling procedure at each generation for a population of size n = 3m is $m\cdot(5m - 1)/2$, while in the total reshuffling operation the total number of crossovers is $0.6\cdot(3m)(3m-1)/2$ for a reproduction rate of 0.6. These two numbers are not very different even for large n, since the ratio $[m(5m-1)/2]/[3m(3m-1)] = (5m-1)/[3(3m-1)]$ tends slowly to a limit 5/9 (=0.556) as n increases.
Thus the probable improvement in fitness value at each generation in both cases should be comparable. However, the reshuffling requires the process to start over from the very beginning. This means that our partial reshuffling procedure normally requires much less time to reach the same degree of improvement.

4. COMPUTATIONAL RESULTS.

The modified genetic algorithm described previously was implemented with the following settings:
The population size is 100 chromosomes.
A two-point crossover and a uniform mutation (set to 0.7) are employed. The crossover operation can only be performed between 2 chromosomes that respect the defined overlapping rate, which is set to 0.7.
The reshuffling operation is performed if 200 generations without improvement

The curves represent the result for a representative instance. The case considered is a 5 components, i.e. 31 products-case, the number of modules kept in inventory is set to 20.

Fig. 4 represents the evolution of the best mean assembly time for the different generations obtained with the GA .The first noticeable improvements are due to the partial reshuffling procedure, this can be observed on the curve as the improvements occur every 200 generations (number of generations without improvement between reshufflings).
Fig. 5 represents only the generations that induce an improvement of the best assembly time. The interest of the reshuffling is here also obvious: the improvements resulting of a “normal” generation (crossover and mutation, on the graph generations 3, 4, 6 and 7) lead only to reduce relative improvement in comparison to generations 1, 2, 5, 8 and 9 (reshufflings).

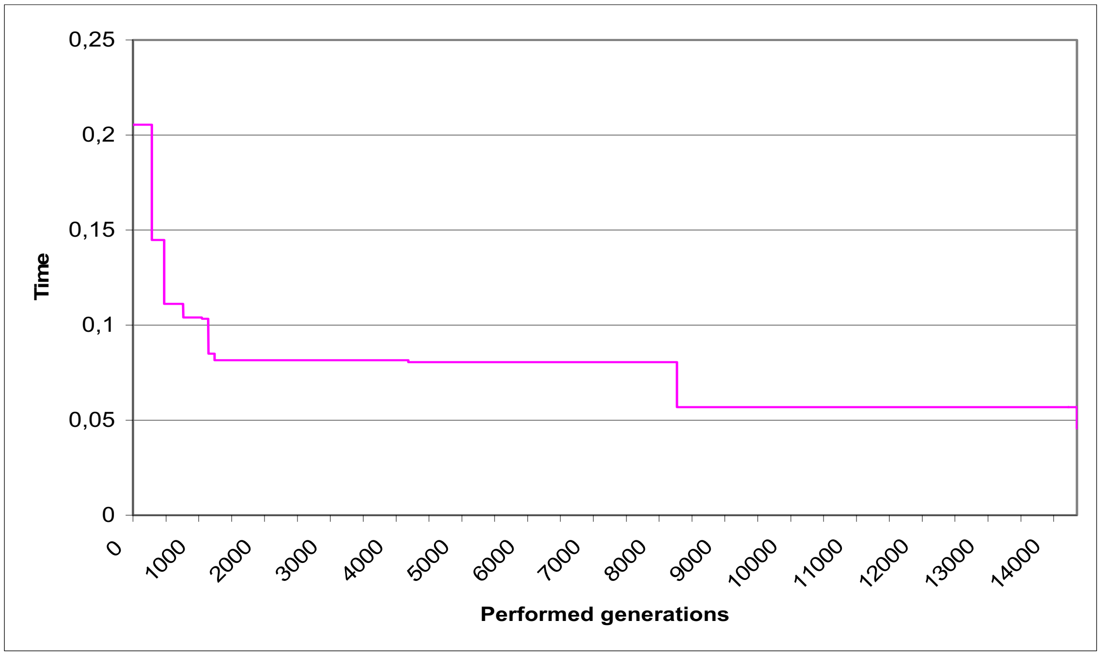

Fig. 4. Best mean assembly time of the different generations.

The mutation rate impacts the number of generations needed to obtain an optimum. This coefficient can therefore be modulated in order to improve the convergence of the GA. For example the mutation rate can be reduced after each generation that improves the best solution. The lowest curve in Figure 5 represents the results obtained for the same instance with a mutation rate *mr* set to 0.7 at the beginning, but modified as follows: at each improving generation *mr = mr*0.9.*

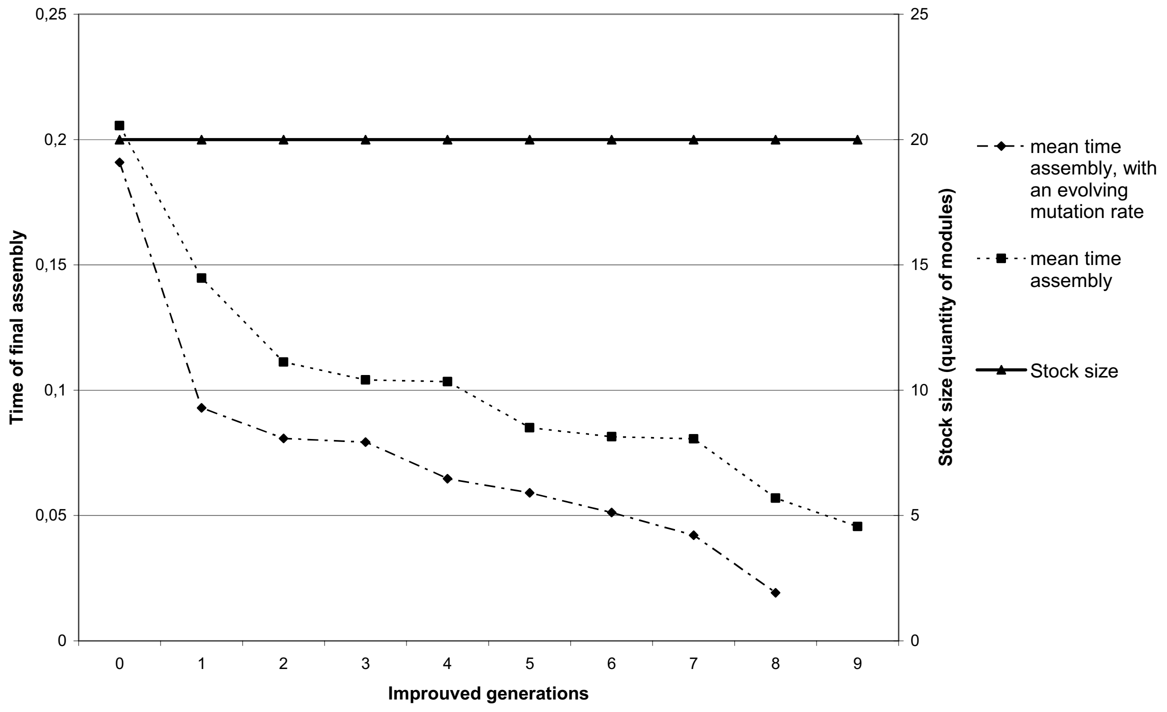

Fig. 5. Evolution of the best mean assembly time, with and without an evolving mutation rate.

The convergence is much better and the algorithm stops at the 8th improving generation significant reduction in mean assembly time.

5. CONCLUSION

The problem treated deals with the selection of a fixed number of modules for the assembly of diversified products minimizing the delay of final assembly.

When introducing the methodology for solving this problem, it was argued in section 3 that the genetic algorithm was naturally adapted and should give good solution if it is properly implemented. This is verified by the experimental results, especially when the modified crossover, the adaptive mutation operation and the partial reshuffling procedure are implemented, the overall improvement has been dramatic with a total of almost 90% reduction of mean assembly time. Further investigation could focus on more elaborate models that minimizes total manufacturing cost or larger models with constraints between modules (inclusion and/or exclusion) and constraints on the production sites (capacities). We expect that similar good experimental results will be obtained. The reports on these will be forthcoming.

ACKNOWLEDGMENTS

The authors wish to acknowledge the support of the Natural Science and Engineering Research Council of Canada (NSERC). The research project has been also supported by the Fonds Québécois de la Recherche sur la Nature et les Technologies (FQRNT).

REFERENCES

Cachon, G. (2003) Supply Chain Management: Design, coordination and operation, vol. 11 of Handbooks in Operations Research and Management Science, chapter Supply chain coordination with contracts, pp. 229–340, Ed. Graves, Steve and de Kok, Ton.

Cheung, B. K-S. (2005). Genetic Algorithm & Other Meta-Heuristics – The Essential Tools for Solving Modern Supply Chain Management Problems. *Chapter of a book entitled ' Successful Strategies in Supply Chain Management*, (Eds.)

C. K. Chan & W.H.J. Lee, pp. 144-173. Idea Group Publishing, PA, USA.

Chvatal V. (1979) A greedy heuristic for the set covering problem, *Mathematics of Operations Research*, vol. 4, no. 3, pp. 233– 235.

da Cunha, C. and Agard, B. (2005) Composition of modules' stock using Simulated Annealing, *6th IEEE International Symposium on Assembly and Task Planning – ISATP 2005*, Montréal, Canada, July 19-21.

Garey M. and Johnson D. (1979) *Computers and Intractability: A guide to the theory of NPcompleteness*, W. H. Freeman and Company, San Francisco.

Johnson D. (1974) Approximation algorithms for combinatorial problems, *Journal of Computer and System Sciences*, vol. 9, pp. 256–278.

Pine B. (1993) *Mass customization: the new frontier in business competition*, Boston MA: Harvard business School Press.

APPENDIX

Algorithm for computing *$NA(P_i;C)$*.

In order to determine *NA(Pi, C)*, the number of assembly operations to manufacture product Pi, from a stock composition C, a greedy algorithm (Chvatal,1979) is employed.

Let m_k be a module from C, Pi is the product to assemble and G is the list of modules to assemble Pi.

Pi \ G represents the list of components from Pi that are not include in the modules from G.

$m_k \subseteq$ Pi \ G if all components from m_k are present in Pi \ G.

Call card{G} the number of modules in G.

The algorithm is then :

```
Algorithm $NA(P_i, C)$
INPUT Product $P_i$, C
OUTPUT card{G}
    $G = \emptyset$
    While $Pi \setminus G \neq \emptyset$
        Select the module $m_k \in C$, $m_k \subseteq P_i \setminus G$, such
            that mk has as much components as
            possible in common with Pi \ G
        Add $m_k$ in G ($G \leftarrow G + \{m_k\}$)
    End While
    Return card{G}
End
```

Let see with an example:

$Pi = A_1A_2A_3A_5A_6$.

$C = \{A_1;A_2;A_3;A_4;A_5;A_6;A_1A_2;A_1A_5;A_5A_6\}$.

The algorithm gives :

0- $G = \emptyset$, $Pi \setminus G = A_1A_2A_3A_5A_6$

1- $G = \{A_1A_2\}$, $Pi \setminus G = A_3A_5A_6$

2- $G = \{A_1A_2;A_5A_6\}$, $Pi \setminus G = A_3$

3- $G = \{A_1A_2;A_5A_6;A_3\}$, $Pi \setminus G = \emptyset$

4- card{G} = 3

The order of the modules in G does not represent any anteriority in the assembly process.

Remark:

(Chvatal,1979)'s algorithm does not give any criterion to select a module between two that are of the same size (same number of components).