International Journal of Production Research
Publication details, including instructions for authors and subscription information:
http://www.informaworld.com/smpp/title~content=t713696255

Composition of module stock for final assembly using an enhanced genetic algorithm

Bruno Agard [a]; Catherine da Cunha [b]; Bernard Cheung [c]
[a] CIRRELT, Département de Mathématiques et de Génie Industriel, École Polytechnique de Montréal, Montréal (Québec), Canada [b] Laboratoire IRCCyN, École Centrale de Nantes, France [c] GERAD, École Polytechnique de Montréal, Montréal (Québec), Canada

First Published:January2009

To cite this Article Agard, Bruno, da Cunha, Catherine and Cheung, Bernard(2009)'Composition of module stock for final assembly using an enhanced genetic algorithm',International Journal of Production Research,47:20,5829 — 5842
To link to this Article: DOI: 10.1080/00207540802161030
URL: http://dx.doi.org/10.1080/00207540802161030

International Journal of Production Research
Vol. 47, No. 20, 15 October 2009, 5829–5842

Taylor & Francis
Taylor & Francis Group

Composition of module stock for final assembly using an enhanced genetic algorithm

Bruno Agard[a]*, Catherine da Cunha[b] and Bernard Cheung[c]

[a]*CIRRELT, Département de Mathématiques et de Génie Industriel, École Polytechnique de Montréal, Montréal (Québec), Canada;* [b]*Laboratoire IRCCyN, École Centrale de Nantes, Nantes Cedex 03, France;* [c]*GERAD, École Polytechnique de Montréal, Montréal (Québec), Canada*

(*Received 13 September 2007; final version received 22 January 2008*)

The paper focuses on modelling and solving a design problem, namely the selection of a set of modules to be manufactured at one or more distant sites and shipped to a proximity site for final assembly subject to time constraints. The problem is modelled as a mathematical one, and solved by an appropriately designed genetic algorithm enhanced with a modified crossover operation, a uniform mutation with adaptive rate and a partial reshuffling procedure. The actual design problem is solved with 17 components. Larger problems may be solved without modifying the modelling steps, although they may require variation in terms of processing time, depending on the constraints that exist between the components.

Keywords: product design; productivity improvement; data mining

1. Introduction

The current competitive environment is such that customers have at their disposal many different products to satisfy their personal requirements. For companies, the challenge is to meet the customers' requirements as closely as possible, with a view to selling them their products. The company has to provide the precise product at the precise place and time at which it is needed (Pine 1993).

For a company, broad commercial diversity is viable if it is supported by appropriate decision making, and various philosophies, methods and tools are now available to help in this process. These include mass customisation, modularity, commonality, product platforms, product families, and so on (Jose and Tollenaere 2005, Simpson *et al.* 2007).

Moreover, because of globalisation, the company may have to take into account different production sites, which may be located anywhere in the world, with different production capacities and costs and different levels of reactivity.

In a Business to Business context (B2B), when the customer is another firm, the question of diversity management acquires another dimension. Indeed, the contracts linking the good manufacturer (referred to as the contractor) and the 'customer' often specifies tight due dates.

*Corresponding author. Email: bruno.agard@polymtl.ca

ISSN 0020–7543 print/ISSN 1366–588X online

DOI: 10.1080/00207540802161030
http://www.informaworld.com

In the automotive industry the notion of product specification has a special meaning since upstream substitution has been much used in the last decades: cars were equipped with options that the car buyer did not specifically order. Those options were then either 'offered to the client' or hidden/destroyed before car delivery. The cost of these 'extras' was then shared between the car manufacturer and the automotive supplier.

Consider the following supply chain: the contractor provides one product for each customer; each product corresponding to that customer's requirements. The contractor's assembly plant is typically an assembly line with a fixed tack time. Since all products on the assembly line are potentially different,[1] the contractor requires from its suppliers a delivery that corresponds exactly to its own production schedule. The fixed tack time then becomes a time constraint for the supplier.

For the suppliers, the issue is to provide the precise sub-assembly (module) required, at the precise place and time it is required. The supplier's structure is made up of different production sites located in different areas around the world (see Figure 1). The supplier's production policy is that the distant site(s) (with cheaper manufacturing costs) assemble(s) components into units called modules, which are shipped to the supplier's stock on the proximity site in order to be assembled in a final operation according to the wishes of each customer, respecting a fixed maximum time of delivery for each product. This structure enables synchronous delivery: the correct products have to be delivered at the right time, but also in the required order (Alford *et al*. 2000, Bernier and Frein 2004, Gusikhim *et al*. 2008).

To clarify the notion of distance and proximity, the term 'supplier park' is used in the automotive sector. This notion describes the strategic choice of an automotive supplier to dedicate a facility to a given customer (a car manufacturer). The definition given by Howard *et al*. (2006) is chosen ''A concentration of dedicated production, assembly, sequencing or warehousing facilities run by suppliers or a third party in close proximity (i.e. within 3 km) to the OEM plant.'' In this article the considered proximity/nearby site belongs to one of these supplier parks.

In such a context, modular design and delayed product differentiation may be advantageously combined in an assembly-to-order policy. Assemble to order makes it possible to provide a large number of final products from a limited number of

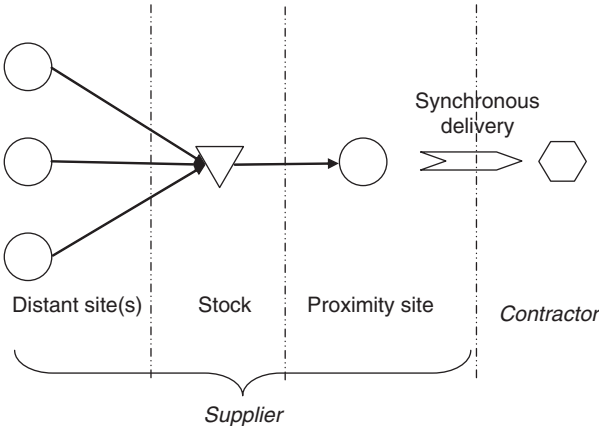

Figure 1. Industrial configuration.

modules (Starr 1965). Each module may be manufactured at a different distant site and shipped to be assembled according to the customer's order requirements when an order is received.

Filling an order within the prescribed time is a major issue (Cachon 2003), and a significant impact on that time is the structure of the supply chain. This article exploits the performance of a genetic algorithm (GA) for the selection of a module composition (a set of modules) that minimises the mean final assembly time for the supplier, in a given supply chain.

In Section 2, the problem is modelled mathematically. Section 3 deals with solving the problem, including encoding the problem with a GA and describing various enhancement procedures (modified crossover operator, uniform mutation with an adaptive rate and a partial reshuffling procedure). These enhancements, with regards to conventional GA (Holland 1975), aim at reducing the redundancy and enlarging the population size. Computational results are provided in Section 4. Section 5 concludes the paper.

2. Problem modelling

The problem described above and presented in Figure 1 is modelled as follows. Consider the following notation: a_i is a component, $i \in [1;n]$, and n is the number of components. The final products P_i may be composed of any combination of the different initial components. In Figure 2, product $P_1 = a_1$ contains only component a_1, product $P_5 = a_1 a_2$ contains both components a_1 and a_2, and so on.

At the distant site(s), the components are combined to provide M modules. M depends on the stock capacity at the proximity site, and is actually an input for the problem. The final assembly realised on the proximity site enables us to obtain every final product from the M modules.

This model is particularly adapted to describe products that can be combined, like do-it-yourself kits (the assembly operation is then only a global packing operation) or

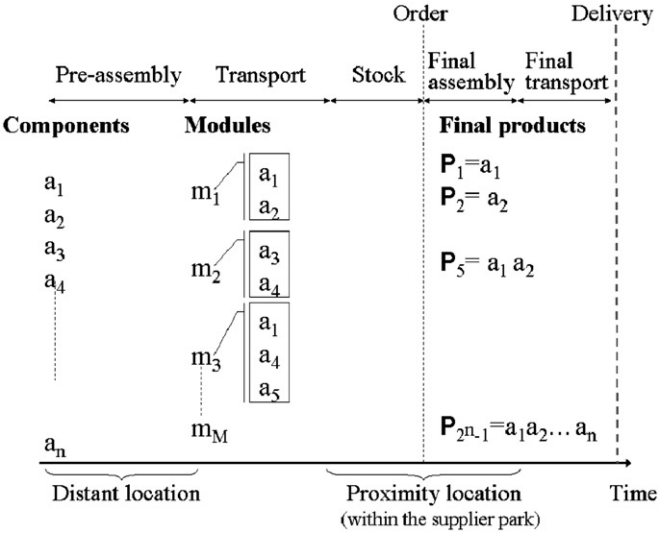

Figure 2. Problem modelling.

electrical wire harnesses. For products that share a common structure, other methods should be used, for example product family design using platforms. See Du *et al.* (2001) and Jiao and Tseng (1999) for implementation methodologies.

An electrical wire harness is a set of wires and connectors (Figure 3). Electrical wire harnesses are used in many products (cars, trucks, airplanes, helicopters, ATM, robots, etc.); they permit the transmission of power and information between different parts (devices, lights, engine, dashboard, etc.) of those products. A subset of wires may be added or removed when options are included or not in the product. For example, a car with an electric sunroof will have a supplementary subset of wires and connectors to activate the sunroof; this supplementary subset of cables will simply be added to the wire harness dedicated to that specific car. To reduce lead time and manufacturing costs, some 'packages' (called modules) are manufactured in advance. It could be the case for the electrical sunroof that the subset of required wires are prepared in advance, and when a car has an electrical sunroof, the module has simply to be joined.

If there are no exclusive or inclusive constraints 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. In real cases, some constraints may exist between components. For example, for a component to be able to satisfy its functionality, it may require another component. This is an implication constraint of the type *Component* $A \Rightarrow$ *Component* B. Also, some components may be alternatives, and therefore it is not possible to use them simultaneously in the same product. This is an example of an exclusion constraint of the type *Component* $A \Rightarrow NOT$(*Component* B). The existence of constraints between components reduces the search space, as it reduces the number of products offered for sale and the number of possible module compositions C. Section 4.2 provides numerical results considering these kind of constraints.

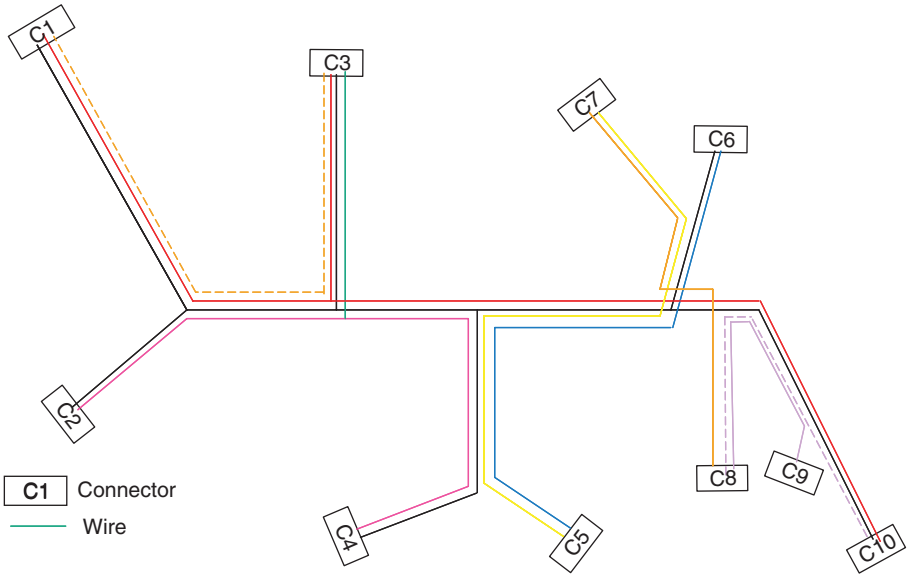

Figure 3. Electrical wire harness.

Each product P_i may more or less satisfy customer requirements, and therefore has a different probability of being sold. Let $p(P_i)$ be the probability that there is a demand for product $P_i \in \mathbf{P}$.

The final demand can be summarised by the vector $D = (p(P_1)p(P_2)\ldots p(P^n_{2-1}))$. The probability of each product i being sold ($p(P_i)$) comes from an analysis of historical sales and projections from the marketing department, and is an input of the problem.

Let C be a module composition; a module composition is a set of modules m_i that is manufactured at the distant site(s) and then shipped to the supplier park for final assembly. Note that $|C|$ represents the number of modules in the module composition. Note also that $T_D(C)$ is the mean time to assemble demand D from the stock composition C.

A supplementary constraint is that no product can have any extra components or any doublet (only different options are assembled). The problem with extra components (upward substitution) is a set-covering problem, and that problem may be found in Briant and Naddef (2004).

A stock composition C is evaluated using two criteria:

- the number of selected modules, $|C|$, and
- the mean assembly time, $T_D(C)$.

For a fixed number of selected modules, $|C|$ equals M (due to the proximity site capacity), and the optimisation concerns the minimisation of the mean assembly time $T_D(C)$.

For a demand D, the problem becomes the selection of a module composition (a set of modules) of size M that minimises the mean assembly time and allows the production of any product in $\mathbf{P}$. For simplification, the assembly time of a final product is considered to be proportional to the number of assembly operations (two assembly operations need an assembly time of two time units). The mean assembly time is then assimilated into the expected value of the number of assembly operations. This assumption reflects the fact that the assembly of two modules involves the same level of difficulty, whatever the modules. For instance, a final product obtained by assembling two modules will have an assembly time of 1 unit, while a final product obtained by assembling four modules will have an assembly time of three units.

Note that $N(P_i, C)$ is the number of assembly operations required to manufacture the product P_i starting from a module composition C. $N(P_i, C)$ is impacted by the presence in C of a module (or many modules) which may be included in P_i. If such a module is present, it will reduce $N(P_i, C)$, because some operations have been performed at the distant site(s).

Then,

$$T_D(C) = \sum_i p(P_i) \times N(P_i, C).$$

To obtain $N(P_i, C)$, an exact set-covering problem has to be solved which gives the number of assembly operations needed to obtain P_i from C. This problem is known to be NP-hard (Karp 1972, Garey and Johnson 1979). A greedy algorithm was chosen to evaluate $N(P_i, C)$ (Johnson 1974, Chvatal 1979). This approximation enables a rapid solution. The algorithm for computing $N(P_i, C)$ is provided in Appendix A.

Research to achieve near-optimal solutions for this problem has been intense since the development of this simple greedy heuristic. Nevertheless, a proof that the heuristic cannot be easily approximated (Lund and Yannakakis 1994) prompted us to use this 'early days' heuristic.

The problem is then the following:

$$\text{Min } T_D(C) = \sum_i p(P_i) \times N(P_i, C), \tag{1}$$

s.t.

$$|C| = M. \tag{2}$$

Note that the number of possible module compositions increases almost exponentially as the number n of components increases. This renders the problem intractable with conventional methods, although a good meta-heuristic method may help give a good approximate solution.

To solve this problem, a two-step method has to be used: first, find a composition that verifies constraint (2), and then evaluate the number of assemblies induced, for each product P_i. Since this second step requires the resolution of a NP-hard problem (number of assembly operations needed), the pertinence of the two-step procedure depends on the efficiency of the first step, i.e. composition selection. Fast heuristics based on the probabilities of using the modules were used to solve this problem (da Cunha 2004); the results could be used as a good initial population for a meta-heuristic. The use of a meta-heuristic seems an interesting method for improving the result. Furthermore, the structure of the problem led us towards a GA: a composition is described by the presence or absence of the various possible modules, and can be easily modelled by a chromosome, which is described by the presence or absence of the possible genes.

3. Problem solving using a genetic algorithm

The problem modelled in the previous section is solved using a GA. Here, the application of genetic operators can provide an efficient search of all possible module compositions for the one that gives the shortest mean assembly time, as the fast and accurate algorithm employed for computing $N(P_i, C)$ makes really fast fitness evaluations possible (i.e. evaluations of $T_D(C)$). The encoding is presented in subsection 3.1. Subsection 3.2 presents enhancements for the GA search scheme.

3.1 *Encoding with a GA*

In this section, a modelling of the problem is provided that enables us to encode a GA capable of solving our problem efficiently. Recall that the objective of the algorithm is to find the optimal module composition (the set of modules m_i that minimises the mean assembly time) for a given demand D and a given number of modules kept in stock M.

Regard each chromosome as a module composition C represented by a binary vector, where each digit states the presence or absence of the given module in the composition. The number of digits is equal to the number of modules μ to be considered for possible inclusion in C, $\mu \leq 2^n - 1$ (the maximum value is possible if all the combinations of components a_i can be considered). Let C be a module composition. $C[k] = 1$ when module m_k belongs to the composition, and $C[k] = 0$ otherwise. The number of modules in stock can then be expressed as follows:

$$|C| = \sum_{k=1}^{\mu} C[k].$$

The fitness value of a given composition is defined as

$$\text{Fitness value } (C) = \begin{cases} K - TD(C), & \text{If the composition enables assembly of the entire product portfolio and } |C| = M, \\ 0, & \text{else,} \end{cases}$$

where K is a large-value constant.

As the goal is to minimise the mean assembly time, a greater fitness value means a better performance of the chromosome. A chromosome representing a stock composition with a size greater than M receives a penalty (fitness value 0). The crossover generation chosen here is a two-point crossover, the points being randomly chosen (with a uniform law).

Let us illustrate this operation with an example. Let C1 and C2 be the parents, and C3 and C4 the children (see Figure 4).

$$\begin{aligned} \text{C3}[k] &= \text{C1}[k], \quad 1 \le k < i \quad \text{or} \quad j < k \le 2^n - 1, \\ \text{C3}[k] &= \text{C2}[k], \quad i \le k \le j, \\ \text{C4}[k] &= \text{C2}[k], \quad 1 \le k < i \quad \text{or} \quad j < k \le 2^n - 1, \\ \text{C4}[k] &= \text{C1}[k], \quad i \le k \le j. \end{aligned}$$

3.2 *Enhancement of the GA search scheme*

An enhanced genetic search, introduced by Cheung (2005), with appropriate crossover and mutation operators is considered for improved performance. The details of these modifications are given below.

3.2.1 *Modified uniform crossover operator*

Since all the genes in each chromosome assume a value of only 0 or 1, it can be seen that, in a fairly large proportion of the chromosomes in a randomly generated population, over half (approximately) of its genes will have the same gene values as those at corresponding positions in the optimal chromosome. The crossover operation that exchanges a randomly selected fraction of genes between a pair of chromosomes at their corresponding positions will be the most beneficial. By confining the crossover operations to a pair of distinct chromosomes (e.g. if the overlapping coefficient O_C is less than or equal to a prescribed value O_R, as described later), then

(a) the redundancy in the crossover operation will almost be eliminated, and
(b) the probability of obtaining new solutions that are close to optimal will be greatly enhanced.

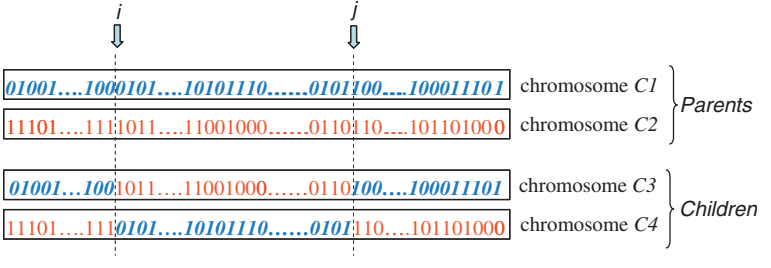

Figure 4. Two-point crossover.

To visualise the effect of the crossover action on a pair of binary chromosomes **a** and **b**, let Z_1 be the set of genes in **a** having gene values common to those of the optimal chromosome at the respective corresponding positions, and let Z_2 be the set of genes in **b** having gene values common to those of the optimal chromosome at the respective corresponding positions. By letting $Y_1 = Z_1 \backslash (Z_1 \cap Z_2)$ and $Y_2 = Z_2 \backslash (Z_1 \cap Z_2)$, this consideration can be confined to the following standard case, as illustrated in Figure 5, since, for fixed i, j and overlap m, this case is unique up to a permutation of their relative positions. Without loss of generality, assume that all the genes of the optimal chromosome have a value equal to one.

Observe that the total number of good solution attributes (i.e. the number of corresponding genes having values identical to those in the optimal solution) is $i+j-m$, where $m = |Z_1 \cap Z_2|$. Now, if the overlap $Z_1 \cap Z_2$ is very small and if the sizes of both Y_1 and Y_2 are large, as in the case where the gene values are binary, then 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 with most of the attributes of the optimal chromosome (see Cheung (2005) for details).

To improve the performance of the two-point crossover so that the crossover operations are limited to chromosome pairs that are not 'too similar', an overlapping measure is defined that evaluates the redundancy of two chromosomes that are to be paired. The overlapping coefficient of two chromosomes C1 and C2 is then defined as

$$O_C(\mathrm{C1}, \mathrm{C2}) = \frac{\mu - \sum_{k=1}^{\mu} |\mathrm{C1}[k] - \mathrm{C2}[k]|}{\mu}.$$

The result will be two identical chromosomes with an overlapping coefficient of 1, while chromosomes with no genes in common will have an overlapping coefficient of 0. The crossover between two chromosomes C1 and C2 will only be authorised if $O_C(\mathrm{C1}, \mathrm{C2}) < O_R$, where O_R is the overlapping rate, set for each implementation.

3.2.2 *Uniform mutation with adaptive rate*

The uniform mutation changes the value of every gene positioned along the length of this chromosome with a probability α. Thus, the uniform mutation randomly changes a fraction α of the total number of gene values in a chromosome. That is, if N is the total number of genes, then αN randomly selected genes will be most likely to be mutated. The mutation operation tends to be disruptive as well as constructive, and an appropriate rate of uniform mutation will give the chromosomes to be mutated a better chance of having a higher 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, and only a few minor changes will be necessary to transform them into the optimal one. Thus, with this scenario, the mutation rate should be adaptively reduced.

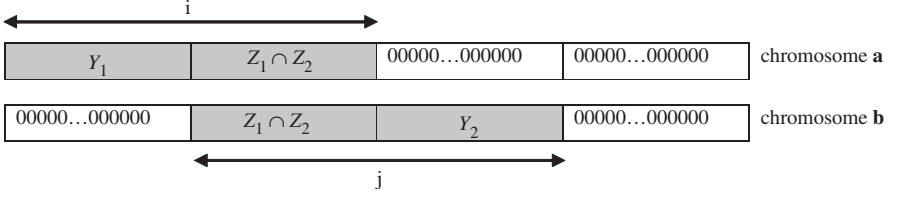

Figure 5. Modified crossover operator.

3.2.3 *Partial reshuffling procedure*

Our procedure takes advantage of the fact that a considerable number of the attributes of the optimal chromosome might have been acquired previously by other chromosomes throughout the process of reproduction. Those chromosomes, with good solution potential and that have been shown to be incapable of any further improvement through extended operations in existing populations, should not be discarded, but be allowed to recombine with newly generated chromosomes for possible new improvements. Clearly, this reshuffling procedure generates a considerable time saving relative to the usual procedure, which requires an actual restart from the very beginning. The details of this procedure are given below.

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

(1) Generate n new strings $\{a_1, a_2, \ldots, a_n\}$ that are different from all the chromosomes in the original population, and arrange them in descending order of fitness.
(2) Otherwise, cross over a_j with all the strings 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 more (say about 80%) of the less well-fitted original chromosomes have been replaced.

Assuming that the generation process is random, and that the probability of obtaining improved new offspring when a mature chromosome crosses over with a new chromosome is approximately the same as that when two new chromosomes cross over with one another, 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 towards a limit of 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 be restarted 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

To measure the performance of the algorithm, the procedure was coded in C++ and tested on various instances. Two kinds of problem are considered in the computational experiments below: first, small problems without constraints in Section 4.1, and, second, larger problems with constraints in Section 4.2.

4.1 *Small-size example*

The modified GA described previously was implemented with the following settings.

- Five components.
- Population size of 100 chromosomes.

- A two-point crossover and a uniform mutation (set to 0.7).
- The crossover operation is performed only between two chromosomes that respect the defined overlapping rate, which is set to 0.5.
- The reshuffling operation is performed if 100 generations do not lead to an improvement.
- The number of possible modules corresponds to all possible combinations of the components: $\mu = 2^n - 1 = 31$.

The curves in Figures 6 and 7 represent the results for this example. The case considered is a five-component algorithm, i.e. a 31-product case, and the number of modules kept in stock is set to 20. In Figure 5, for each generation the mean final assembly

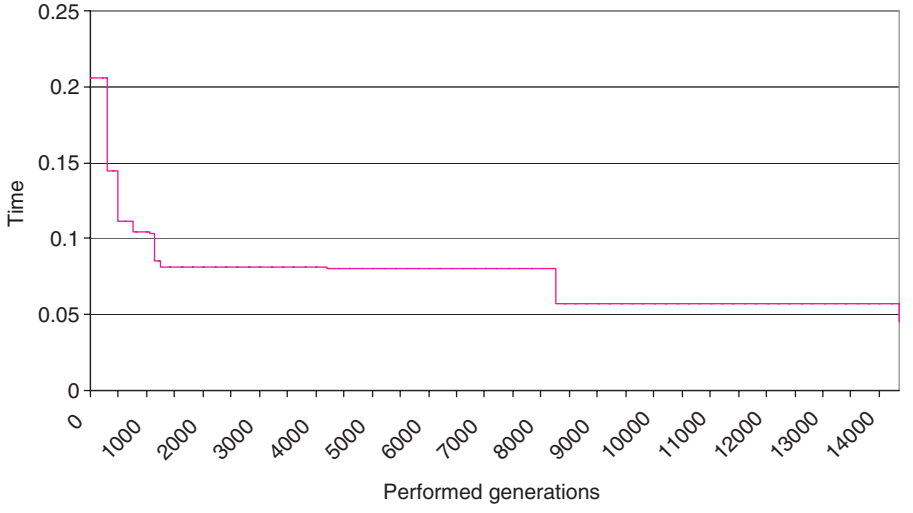

Figure 6. The best mean assembly time of the various generations.

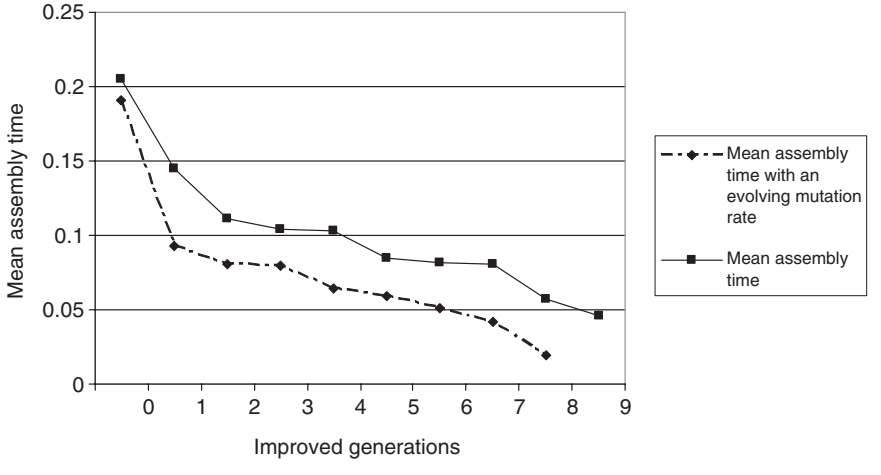

Figure 7. Evolution of the best mean assembly time, with and without an evolving mutation rate.

time is given. Figure 6 uses the same data, but shows only the improved generations. The time is set such that one assembly operation takes one time unit.

These first results show that a GA with the proposed enhancements is highly appropriate for solving this optimisation problem. The best mean assembly time decreases dramatically as a function of the number of generations performed, the first 2000 generations performing 80% of the overall optimisation.

The optimisation enabled a decrease in the best mean assembly time from 0.21 to 0.06 units of time, which represents a 70% improvement. Such a high-level of improvement is possible because 20 modules were selected to manufacture 31 final products. With a smaller stock size, the improvement may not be so significant.

The partial reshuffling procedure enables us to find a stock composition giving a best mean assembly of 0.02 units of time at the eighth generation, which represents an almost 90% improvement (compared with the 0.21 obtained initially).

The first noticeable improvements are a product of the partial reshuffling procedure. This can be observed from the curve, as the improvements occur every 100 generations (number of generations without improvement between reshufflings). Note that many subsequent generations do not lead to any improvement in the best assembly time. Figure 7 presents only the generations that permit an improvement in the best assembly time.

The advantage of the reshuffling procedure is obvious here: the improvements resulting from a 'normal' generation (crossover and mutation, generations 3, 4, 6 and 7 on the graph) only lead to a reduction in the relative improvement in comparison with generations 1, 2, 5, 8 and 9 (reshufflings).

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 7 represents the results obtained for the same instance with the mutation rate α set to 0.7 at the start, but modified at each improving generation by $\alpha = 0.9 \times \alpha$. The convergence is much better and the algorithm stops at the eighth improving generation, resulting in a significant reduction in the mean assembly time.

4.2 *Real-world example with constraints*

As the results were promising with the small example, instances more representative of industrial reality were computed. The results presented here represent an instance of 17 components (i.e. 131,071 potential products). The number of modules kept in stock is set to 50 (this is a relatively small number compared with the number of products). The exclusion and implication constraints between components are introduced by a pre-processing method that removes any forbidden modules from C, so $\mu < 2^n - 1$. Only valid modules are available in C, and then the selected composition will respect the constraints.

The settings for the algorithm were the same as those for the small example (100 chromosomes, two-point crossover, a fixed uniform mutation set to 0.7, an overlapping rate equal to 0.5, reshuffling after 100 generations without any improvement, and an adaptive mutation rate α set to 0.7 at the start and $\alpha = 0.9 \times \alpha$ after each improved generation). The results are presented in Figure 8. Behaviour similar to that of Figure 7 is observed. Nevertheless, because of the size of the example, the number of iterations was

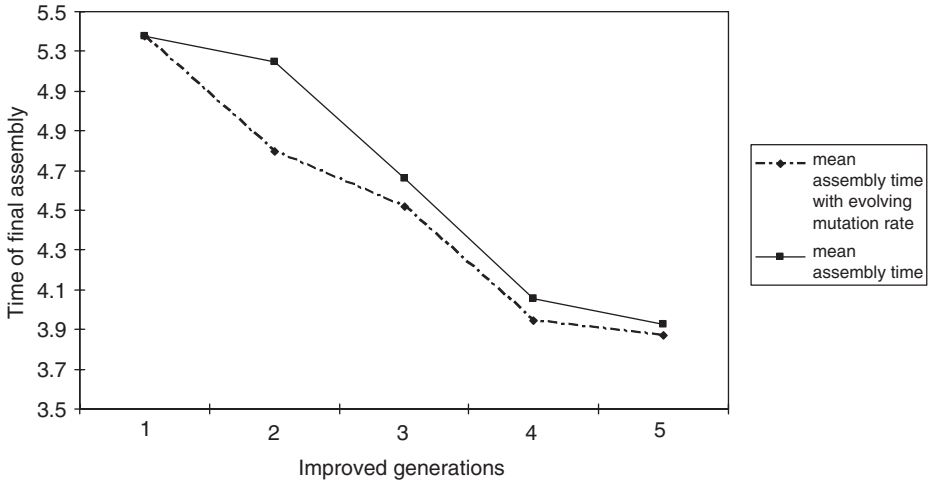

Figure 8. Evolution of the best mean assembly time, with and without an evolving mutation rate (real-sized example).

limited to 2000 generations in the simulation. Figure 8 shows that the method is still efficient for solving larger problems, and supports the inclusion and exclusion constraints in the description of the modules. The relative improvement may vary for a number of possible reasons other than stock size (30% in Figure 8): (1) 'interesting' candidate modules for the optimisation may have been forbidden by the constraints (the search space is smaller); (2) the number of modules (stock size) is an influential parameter: the greater the number of modules, the better the improvement (if there are as many modules as the number of products, the mean assembly time is 0!), but companies cannot handle too many modules (because of cost and space constraints at the proximity site, for example); and (3) the quality of the chromosomes in the initial population.

5. Conclusion

The problem addressed here deals with the selection of a fixed number of modules for the assembly of diversified products to minimise the time taken in final assembly. When introducing the methodology for solving this problem, it was argued in Section 3 that the GA was adapted and should give a 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. In this case, the overall improvement was dramatic, with a total reduction in mean assembly time of almost 90% for the small example. Tests performed with larger examples make it possible to validate the efficiency of the proposed algorithm for real-sized problems with constraints. The use of an adaptive mutation rate associated with the partial reshuffling procedure enables acceleration of the convergence process, which in turn affects the actual performance of the method.

Current modelling and solution methods enable us to consider larger problems (up to 17 components, i.e. 131,701 potential final products). Larger problems can be solved if

constraints are considered, otherwise the search space will be too large and computational time will increase dramatically.

Stock size is an important parameter of the problem and deserves further investigation. The initial chromosomes may be computed thanks to a heuristic to start with better generations and speed up the resolution. Also, further investigation could focus on more elaborate models that minimise total manufacturing cost (including transportation costs) and constraints on the production sites (capacities). Similar good experimental results are expected.

Acknowledgements

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

Note

1. Up to 7 million different wire harnesses can be demanded for a single middle range car (Agard and Tollenaere 2003).

References

Agard, B. and Tollenaere, M., 2003. Design of wire harnesses for mass customization. *In*: G. Gogu, *et al*., eds. *Recent advances in integrated design and manufacturing in mechanical engineering*. Dordtrecht: Kluwer Academic, 53–62.

Alford, D., Sackett, P., and Nelder, G., 2000. Mass customization—an automotive perspective. *International Journal of Production Economics*, 65, 99–110.

Bernier, V. and Frein, Y., 2004. Local scheduling problems submitted to global FIFO processing constraints. *International Journal of Production Research*, 42 (8), 1483–1503.

Briant, O. and Naddef, D., 2004. The optimal diversity management problem. *Operations Research*, 52 (4), 515–526.

Cachon, G., 2003. Supply chain coordination with contracts. *In*: S. Graves and T. de Kok, eds. *Handbooks in operations research and management science*. North Holland: Elsevier, 229–340.

Cheung, B.K-S., 2005. Genetic algorithm and other meta-heuristics—the essential tools for solving modern supply chain management problems. *In*: C.K. Chan and W.H.J. Lee, eds. *Successful strategies in supply chain management*. PA: Idea Group Publishing, 144–173.

Chvatal, V., 1979. A greedy heuristic for the set covering problem. *Mathematics of Operations Research*, 4 (3), 233–235.

Da Cunha, C., 2004. *Definition and inventory management of semi-finished products in an ATO context*. Thesis (PhD). Institut National Polytechnique de Grenoble.

Du, X., Jiao, J., and Tseng, M., 2001. Architecture of product family: fundamentals and methodology. *Concurrent Engineering: Research and Application*, 9 (4), 309–325.

Garey, M. and Johnson, D., 1979. *Computers and intractability: A guide to the theory of NP completeness*. San Francisco: W. H. Freeman.

Gusikhim, O., Caprihan, R., and Stecke, K., 2008. Least in-sequence probability heuristic for mixed-volume production lines. *International Journal of Production Research*, 46 (3), 647–673.

Holland, J., 1975. *Adaptation in natural and artificial systems*. Ann Arbor: University of Michigan Press.

Howard, M., Miemczyk, J., and Graves, A., 2006. Automotive supplier parks: an imperative for build-to-order? *Journal of Purchasing and Supply Management*, 12 (2), 91–104.

Jiao, J. and Tseng, M., 1999. A methodology of developing product family architecture for mass customization. *Journal of Intelligent Manufacturing*, 10 (1), 3–20.

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

Jose, A. and Tollenaere, M., 2005. Modular and platform methods for product family design: literature analysis. *Journal of Intelligent Manufacturing*, 16 (3), 373–392.

Karp, R., 1972. Reducibility among combinatorial problems. *In*: R. Miller and J. Thatcher, eds. *Complexity of computer computations*. New York: Plenum Press, 85–103.

Lund, C. and Yannakakis, M., 1994. On the hardness of approximating minimization problems. *Journal of the ACM*, 41 (5), 960–981.

Pine, B., 1993. *Mass customization: The new frontier in business competition*. Boston, MA: Harvard Business School Press.

Simpson, T.W., Siddique, Z., and Jiao, R.J., eds., 2007. *Product platform and product family design: Methods and applications*. New York: Springer.

Starr, M., 1965. Modular production – a new concept. *Harvard Business Review*, Nov-Dec, 131–142.

Appendix A: Algorithm for computing $N(P_i, C)$

In order to determine $N(P_i, C)$, the number of assembly operations required to manufacture product P_i from stock composition C, a greedy algorithm (Chvatal 1979) is employed. Let m_k be a module from C, P_i is the product to be assembled and G is the list of modules required to assemble P_i. $P_i \backslash G$ represents the list of components from P_i that are not included in the modules from G. $m_k \subseteq P_i \backslash G$ if all components from m_k are present in $P_i \backslash G$. Let card$\{G\}$ be the number of modules in G. The algorithm is then as follows:

Algorithmis $N(P_i, C)$
INPUT Product P_i, C
OUTPUT card$\{G\}$
  $G=\varnothing$
  While $P_i \backslash G \neq \varnothing$
    Select the module $m_k \in$C, $m_k \subseteq P_i \backslash G$, such that m_k has as many components as possible in common with $P_i \backslash G$
    Add m_k in G ($G \leftarrow G+\{m_k\}$)
  End While
  Return card$\{G\}$
End

Let us illustrate this with an example:

$$P_i = a_1a_2a_3a_5a_6,$$
$$C = \{a_1; a_2; a_3; a_4; a_5; a_6; a_1a_2; a_1a_5; a_5a_6\}.$$

The algorithm gives

0—$G=\varnothing$, $P_i \backslash G=a_1a_2a_3a_5a_6$
1—$G=\{a_1a_2\}$, $P_i \backslash G=a_3a_5a_6$
2—$G=\{a_1a_2;a_5a_6\}$, $P_i \backslash G=a_3$
3—$G=\{a_1a_2;a_5a_6;a_3\}$, $P_i \backslash G=\varnothing$
4—card$\{G\}=3$

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

Remark: Chvatal's (1979) algorithm does not give any criterion for selecting a module between two of the same size (same number of components).