Int. J. Production Economics 117 (2009) 389–401

Contents lists available at ScienceDirect

Int. J. Production Economics

journal homepage: www.elsevier.com/locate/ijpe

A simulated annealing method based on a clustering approach to determine bills of materials for a large product family

Bruno Agard [a,*], Bernard Penz [b]

[a] CIRRELT, Département de Mathématiques et Génie Industriel, École Polytechnique de Montréal, C.P. 6079, succ. Centre-Ville, Montréal (Québec), Canada H3C 3A7
[b] G-SCOP, Grenoble INP–CNRS–UJF, 46 avenue Félix-Viallet, 38031 Grenoble Cedex 1, France

ARTICLE INFO

Article history:
Received 3 December 2007
Accepted 4 December 2008
Available online 10 December 2008

Keywords:
Product design
Bill of materials
Optimization
Clustering
Simulated annealing
Mass customization

ABSTRACT

Defining an efficient bill of materials for a family of complex products is a real challenge for companies, largely because of the diversity they offer to consumers. To remain competitive, their products are available with many variants and options, which leads to a difficult question: "How can a bill of materials be defined so that all possible products can be built efficiently?" One way to answer it is to define a set of components (called *modules*), each of which contains a set of primary functions. An individual product is then built by combining selected modules. This industrial problem leads, in turn, to the complex optimization problem we study in this paper, which we propose to solve using a simulated annealing method based on a clustering approach.

1. Introduction

Confronted with a broad array of products on the market today, customers are becoming increasingly demanding. Every product sold needs to meet customer requirements exactly in terms of functionality and be offered at the lowest possible price. As a result, markets are becoming more and more segmented. In order to increase their market share without expensive product proliferation, companies focus on "mass customization" (Pine, 1993). Mass customization deals with a large product portfolio, flexible manufacturing systems and an extensive supply chain.

A product family is composed of similar products which differ in terms of characteristics like options and variants. Take, for example, a personal computer. The basic model contains few options and cheap components to minimize the retail price. Subsequently, this basic model can be modified by adding, say, a graphics card, a DVD player, a Wi-Fi card, and so on. It can also be improved by modifying the quality of the components, for example including a more sophisticated processor or better cards (Lee and Tang, 1997). This industrial problem arises in other domains as well, like the car industry for electrical wire harnesses (Briant and Naddef, 2004).

If the customer is unwilling to wait for a product, the strategy is to keep various products in stock. In this case, standardization can be an effective tool in product diversity management. The objective in this case is to stock products which include options to meet various customer requirements. The problem then becomes the selection of a minimum set of relevant standardized products (Briant and Naddef, 2004). This strategy makes it possible to serve the customer immediately upon receipt of an order. Unfortunately, the number of finished goods may be too large, which results in unreasonable storage costs.

If the diversity and storage costs are too high, a second, extreme strategy is to produce only when an order is received from a customer. In this case, the lead time to produce the item may be too great to satisfy the customer.

* Corresponding author.
E-mail address: bruno.agard@polymtl.ca (B. Agard).

0925-5273/$ - see front matter Crown Copyright © 2008 Published by Elsevier B.V. All rights reserved.
doi:10.1016/j.ijpe.2008.12.004

An intermediate solution would be to finish the products from pre-manufactured components when an order is received. This solution involves producing some parts of the finished product, called modules, to be kept in stock and assembling them when an order is received. Modules can be manufactured in countries where production costs are low, and the final product can be assembled close to the market in order to be responsive to the demand.

The design of product families has received a great deal of attention in the literature. It is often considered to be an integral design element in mass customization (Pine, 1993). The goal is to provide broad product diversity with a rationalized product structure (Simpson et al., 2006).

A rationalized product structure may be achieved with a dedicated product platform. Simpson et al. (2006) propose various methods for designing such a platform. It is selected based on different indicators, such as commonality, product costs, position of the point of differentiation, and so on. A recent overview of the design of product families may be found in Jiao et al. (2007a) and JoseFlores and Tollenaere (2005).

Product families take advantage of modular design (Kusiak, 1999). Modular components are shared between different products on the same platform or between product platforms. Different types of modules are currently used in industry, e.g. "vanilla" boxes, which are assembled from basic components only (Swaminathan and Tayur, 1998) or "cadillac" boxes, which are oversized modules that can be disassembled or used for downward substitution (Rao et al., 2004). The cost of offering a large portfolio should be balanced against the gain resulting from customer satisfaction. When a generic bill of materials is available, Jiao et al. (2007b) propose a generic genetic algorithm to maximize the customer-perceived cost/benefit ratio of offering a design alternative under certain constraints.

In an assemble-to-order (ATO) context, a product is assembled from stored components when an order is received. Furthermore, a short delivery time must be respected to satisfy the client, which is often shorter than the total time required to manufacture the products from elementary components. ATO takes advantage of delayed product differentiation, modularity in products and processes, as well as standardization (Lee and Tang, 1997).

In some cases, oversized (standardized) products, i.e. products containing functions not demanded by the client, cannot be tolerated. Consequently, it is necessary to provide the exact products that correspond specifically to the requirements of each customer. It is then necessary to take into account the way the final products will be assembled to define the set of necessary pre-assembled components to stock (Agard and Kusiak, 2004). The question is then to define the best set of modular components that enable the assembly of any kind of final product within a predefined period of time. Some attempts have been made to solve that problem by considering minimizing the time of final assembly for a fixed and predefined number of modules with a genetic algorithm (Agard et al., 2006) or, on relatively small problems, using a simulated annealing method (da Cunha and Agard, 2005). These techniques permit to verify if the lead time wanted by the client is respected for a given set of stored modules.

This novel approach consists of providing a quantitative decision aided tool for the design of a large product family. The model consists of the minimization of global costs, including management and production costs, in a distributed supply chain. This model improves previous researches since it considers in the same modeling both the design of the product family and the design of the supply chain. Our model is close to Lamothe et al. (2006) that propose an integrated selection of the product and the required facilities in the supply chain, from a given generic bill of materials. The main difference is that we propose a method that builds the bill of material and the supply chain simultaneously during the optimization process.

The problem considered here is to define the best set of modules permitting a final assembly within a predefined time. That novel formulation and the methodology proposed to solve it allows us to consider much larger problems (13 functions, 8000 modules and 500 products—bigger problems will necessitate more computing time). This new proposition is now adapted for real industrial problems. The challenge for companies is to define what modules they build for stock, and the bill of materials of the finished goods. We attempt to solve this difficulty here using a simulated annealing method with a cluster algorithm to reduce the computation time at each step.

In Section 2, the optimization problem is described more precisely. An Integer Linear Program of the problem is given in this section and used in subsequent sections of the paper. The simulated annealing algorithm is presented in Section 3 and computational experiments are conducted in Section 4. Finally, concluding remarks and perspectives are proposed (Section 5).

2. An optimization model to determine a bill of materials with a subset of modules of minimum cost

Let us consider a set $\mathscr{P}$ of n products. These products may be ordered by at least one consumer during the life cycle of the product family. A product P_i, $i \in \{1,\ldots,n\}$ is associated with a binary vector p_i of size q representing the function it contains. The k-th component of p_i is 1 if the product contains the function F_k, $k \in \{1,\ldots,q\}$, and 0 otherwise (see Fig. 1).

To build finished products, modules are used. As with products, a module M_j is a binary vector m_j of size q representing the function it contains. A module is then defined in the same way as a finished product. The set $\mathscr{M}$ contains m modules. Constituting the set may be a selection of modules defined by the engineering, or it may contain all the possible modules derived from the whole combinatorics, except the module containing only zeros (see Fig. 1).

The notations are the following:

- $\mathscr{F} = \{F_1,\ldots,F_q\}$: the set of q functions that can appear in both the finished product and the modules;

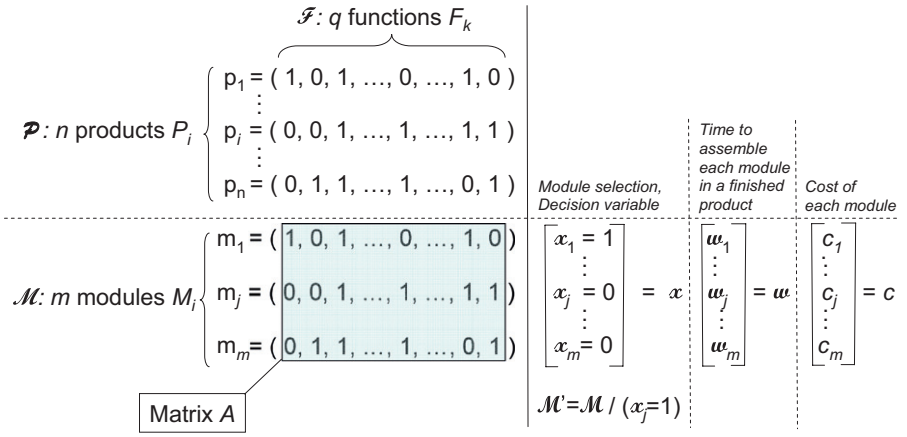

Fig. 1. Representation of parameters and variables.

- $\mathscr{P} = \{P_1, \ldots, P_n\}$: the set of n possible finished products, where p_i is a binary vector of size q;
- $\mathscr{M} = \{M_1, \ldots, M_m\}$: the set of m possible modules, where m_j is a binary vector of size q;
- A: binary matrix of size $q \times m$ formed by all m_j;
- w: vector of size m representing the time spent to assemble each module in a finished product;
- W: maximum assembly time. This time is defined for the assembly operations and do not include previous operations such as the setup of the assembly line, or next operations such as testing, painting or packaging, which all together represent the lead time;
- c: vector of size m representing the fixed cost of modules (for the management of each module);
- q_{min}: minimum number of functions appearing in a product in $\mathscr{P}$;
- q_{max}: maximum number of functions appearing in a product in $\mathscr{P}$.

The problem is now to determine the subset $\mathscr{M}' \in \mathscr{M}$, of minimum cost, such that all products in $\mathscr{P}$ can be built in a constrained time window. The selected modules can be represented by a binary vector x of size m in which 1 means that the corresponding module will be in at least one product. To ensure feasibility, the bill of material y^i of each product P_i must be built. y^i is also a binary vector of size m such that, for each function F_k equals to 1 in p_i, exactly one selected module contains the function F_k, and for each function F_k equals to 0 in p_i, none of the selected modules contains the function. Furthermore, the finished product must be assembled in W or fewer units of time, knowing that a module M_j takes w_j units of time to be assembled in the finished product. The vector w of size m is then the vector of all w_j (see Fig. 1).

The decision variables involved in this problem are as follows:

- x: binary vector of size m such that $x_j = 1$ if the module M_j is used in at least one bill of materials, 0 otherwise. Vector x exactly defines the set $\mathscr{M}'$ of produced modules;
- y^i: binary vector of size m such that $y^i_j = 1$ if the module M_j is used in the bill of materials of product P_i, and 0 otherwise.

Using the previous notations, the described problem can be formulated as follows:

$$\min \quad (cx)$$

$$\text{s.t.} \quad Ay^i = p_i, \quad \forall P_i \tag{1}$$

$$y^i_j \leqslant x_j, \quad \forall P_i \ \forall M_j \tag{2}$$

$$(y^i)^{\mathrm{T}} w \leqslant W, \quad \forall P_i \tag{3}$$

$$x, y^i \in \{0,1\}^m \tag{4}$$

The objective is to minimize the total cost of the selected modules. Vector c could contain any positive real values. The problem is then to minimize the total fixed cost for all modules. In this kind of problem, fixed costs are due to the management of the various references (dedicated tools, adaptation of the production line, specific area in the warehouse, and so on). In a defined product line these costs may be almost similar between modules and then could be all fixed to 1, which leads to a minimization of the number of modules.

Four constraints in the mathematical formulation must be respected:

- Constraint (1) guarantees that, for each product P_i, the bill of materials contains the desired function F_k exactly once, and does not contain an undesired function. This constraint is given in a matrix formulation using matrix A, vectors y^i and p_i as explained in Fig. 1.
- Constraint (2) guarantees that if a module M_j is used in at least one bill of material (at least one y^i is equal to 1), then the module M_j must be selected (M_j belongs to $\mathscr{M}'$).
- Constraint (3) ensures that the final assembly time is less than or equal to the predefined bound W for each product.

- Constraints (4) impose the condition that vectors x and y^i are in $\{0,1\}^m$.

Clearly, this problem is NP-hard in the strong sense, because it contains the well-known Set Partitioning Problem (Garey and Johnson, 1979).

3. A simulated annealing algorithm using a cluster-based approach

This optimization problem cannot be solved by standard optimization software for large instances. As explained previously, the problem is an NP-hard 0–1 optimization problem. The goal is thus to reduce the complexity of the problem in order to solve it approximately. For this, a simulated annealing procedure is proposed.

We explain the general algorithm more precisely in Section 3.1. Then, in Section 3.2, we describe the process of selecting representatives. Stochastic search algorithms are given in Section 3.3. Finally, the way in which the feasibility of a product is tested is described in Section 3.4.

3.1. General scheme of the algorithm

At each iteration t of the method, the main steps are as follows (see Fig. 2): (1) a small set of representatives $\mathscr{P}'_t$ of the products $\mathscr{P}_t$ is generated by a cluster-based heuristic. More precisely, the products found to be not feasible by the current set of chosen modules $\mathscr{M}_{t-1}$ remain in $\mathscr{P}_t$. (2) A set of modules $\mathscr{M}_t$ is generated. This set is optimized (cost minimization by deleting useless modules and adding new ones) by means of the simulated annealing algorithm. The feasibility of products in $\mathscr{P}'_t$ is ensured during this phase. (3) A local cleaning removes new modules from $\mathscr{M}_t$ that are not necessary for the set of representatives $\mathscr{P}'_t$. (4) Verification for all products in $\mathscr{P}_t$ if a feasible bill of material exists. Products that do not verify the test are placed in set $\mathscr{P}_{t+1}$ for the next iteration. When $\mathscr{P}_t$ is empty or if the maximum number of iteration (t_{max}) is reached a global cleaning (5) removes all modules in $\mathscr{M}_t$ that are not necessary for the initial set of products $\mathscr{P}_t$.

The principle of the iterative algorithm is the following (see Algorithm 1). A first set of products $\mathscr{P}_1$ is created containing all the products in $\mathscr{P}$. The algorithm `cluster` is used to obtain the set of representatives $\mathscr{P}'_1$. The simulated annealing is launched in order to determine an optimized set of modules ($\mathscr{M}_2$) that permits all products in $\mathscr{P}'_1$ to be built satisfying the constraints. The algorithm continues until $\mathscr{P}_t$ is empty.

Algorithm 1. Bill of material generator.

Define $\mathscr{P}_t$ as the set of products at iteration t that do not have a bill of material.
Define $\mathscr{P}'_t$ as the set of representatives at iteration t.
Define $\mathscr{M}_t$ as the set of selected modules at iteration t.
$\mathscr{P}_0 \leftarrow \mathscr{P}$
$\mathscr{M}_0 \leftarrow \emptyset$
while ($\mathscr{P}_t = \emptyset$ or $t > t_{max}$) **do**
  $t \leftarrow t+1$
  `cluster`($\mathscr{P}_t, \mathscr{P}'_t$)
  `SA`($\mathscr{P}'_t, \mathscr{M}_t, \mathscr{M}_{t+1}$)
  `LocalCleaning`($\mathscr{P}'_t, \mathscr{M}^*_t$)
  `check`($\mathscr{P}_t, \mathscr{M}_{t+1}, \mathscr{P}_{t+1}$)
end while
`GlobalCleaning`($\mathscr{P}_t$)
return $\mathscr{M}_t$

Some unnecessary modules could be present in $\mathscr{M}_t$ after each simulated annealing or following global optimization, which is a union of partial optimizations. The cleaning procedure tests whether or not each module (in $\mathscr{M}^*_t$ for the local procedure and $\mathscr{M}_t$ for the global procedure) could be removed for the set of products in $\mathscr{P}'_t$ (local procedure) or in $\mathscr{P}_t$ (global procedure).

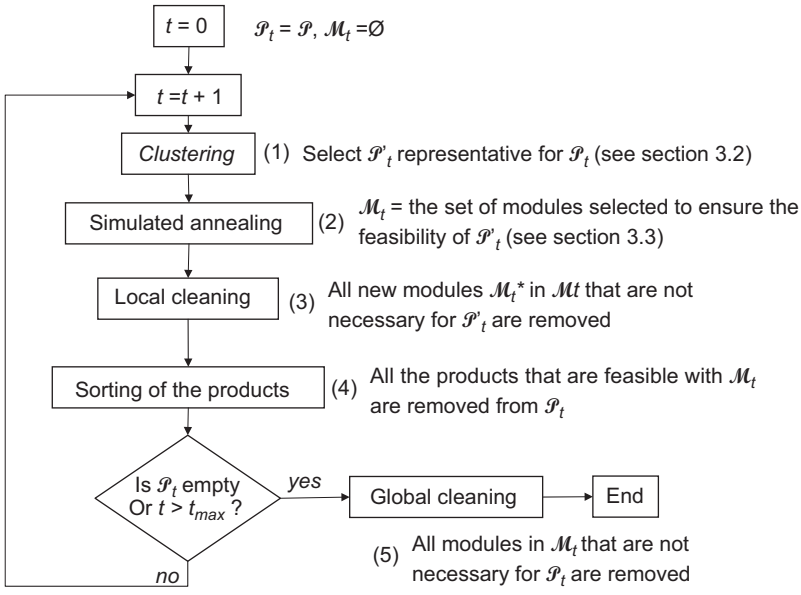

Fig. 2. General methodology.

3.2. Product representatives derived from a cluster-based heuristic

To avoid the difficulty associated with large problems, the set of products $\mathscr{P}_t$, which can be large in certain instances (more than 500 products), is reduced to a new set containing only a few representatives of the whole set.

To obtain these representatives, a k-mean clustering algorithm is used. Readers not familiar with clustering methods may refer to Seber (1984). The k-mean algorithm minimizes the sum, over all clusters, of the distance to the centroid of each cluster. Each centroid is the component-wise median of points in that cluster. The method used is based on Spath (1985), and is a two-phase iterative algorithm. In the latter phase, the distance between two products represents the percentage of data that differ from one element to another, i.e. the Hamming distance. The clusters are denoted C_l with $1 \leqslant l \leqslant n_{cluster}$.

Once the clusters have been obtained, the representatives are the products closest to each centroid. The set $\mathscr{P}_t$ is then replaced by the set $\mathscr{P}'_t$ of representatives, using the method called `cluster(input:`$\mathscr{P}_{in}$`;output:`$\mathscr{P}_{out}$`)`.

3.3. Simulated annealing to determine an optimized set of modules

The goal of this algorithm is to find an optimized set of modules (say $\mathscr{M}_t$) that permits construction of all the products contained in $\mathscr{P}'_t$, satisfying constraints (1)–(4).

The method, called `SA(input:` $n_{departure}, \mathscr{P}_{in}, \mathscr{M}_{in}$`;output:` $\mathscr{M}_{out}$`)`, uses a simulated annealing technique. More information on this now well-known technique is contained in the seminal work of Metropolis et al. (1953), or to the Aarts and Lenstra's (1997) book. The principle of `SA` is to choose a first solution, which we call $\mathscr{M}_1$, represented by a binary vector x_1 of size m and having a value $v(\mathscr{M}_1) = cx_1 + \delta(\mathscr{M}_1)$. The solution could be either feasible or not feasible. If it is not, the value $\delta(\mathscr{M}) = \alpha \times |\textit{unfeasible } P'_i|$ is added to the solution value to ensure that a feasible solution will always be preferred. The value $\delta(\mathscr{M})$ equals 0 when the solution $\mathscr{M}$ is valid. A neighbor $\mathscr{M}_2$ of $\mathscr{M}_1$ is generated using the function `neighbor(input:` $\mathscr{M}_{in}$`;output:` $\mathscr{M}_{out}$`)` and is accepted as the current solution if the value associated with it is strictly better than the value of $\mathscr{M}_1$. Otherwise, it is accepted with a certain probability.

The starting point for `SA` represents each module containing each individual function.

The neighborhood is defined as follows. A fixed number n_{swap} of components is chosen in vector x_1, and for each of them, the value is changed i.e. 0 becomes 1, and 1 becomes 0. Two cases are considered: if the current solution does not permit the assembly of each representative product (meaning that the current solution is not feasible), then the neighbor considers adding modules in the current solution (some 0 become 1), or, if the current solution is feasible, some modules are removed from the current solution (some 1 become 0).

The probability of acceptance of solution x_2 knowing that the current solution is x_1 ($p(x_1 \to x_2)$) is calculated according to a coefficient β which decreases step by step during optimization. When a predefined number of neighbors ($n_{nochange}$) has been explored without any improvement then β is updated:

$$p(x_1 \to x_2) = \beta \tag{5}$$

The optimization procedure stops when no improvements have occurred during the last $n_{nochange}$ iterations, or when the maximum number of iterations $n_{iteration}$ is reached. Then a cleaning procedure (*clean*) removes all the unnecessary modules from the current solution.

Note that in the `SA` method, the generation of x_1 and the neighborhood do not suppress modules in the previous set of modules $\mathscr{M}_{t-1}$. This guarantees that a product $P_i \in \mathscr{P}$ can be made with $\mathscr{M}_t$ if it can be built with modules in $\mathscr{M}_{t-\tau}$ (with $\tau \geqslant 1$). The `SA` algorithm is given in pseudo-code (Algorithm 2).

Algorithm 2. `SA(input:` $n_{departure}, \mathscr{P}_{in}, \mathscr{M}_{in}$`;output:` $\mathscr{M}_{out}$`)`.

Define $\mathscr{M}_{best}$ as the current best module set found during the algorithm execution
Define $\mathscr{M}_{current}$ as the current module set, $\mathscr{M}_{current} = (F_1, F_2, \ldots, F_q)$
for $k = 1$ to $n_{departure}$ **do**
  $\beta = \beta_k$
  $i_{change} = 0$, $i = 0$
  while $i \leqslant n_{iteration}$ and $i_{change} \leqslant n_{change}$ **do**
    $i_{change} = i_{change} + 1$, $i = i + 1$
    if $v(\mathscr{M}_{current}) < v(\mathscr{M}_{best})$ **then**
      $\mathscr{M}_{best} \leftarrow \mathscr{M}_{current}$
      $i_{change} = 0$
    else
      accept $\mathscr{M}_{best} \leftarrow \mathscr{M}_{current}$ with probability $p(x_1 \to x_2) \leqslant \beta$
    end if
    `neighbor(`$\mathscr{M}_{current}, \mathscr{M}_{current}$`)`
  end while
end for
clean
return $\mathscr{M}_{best}$

3.4. An exact algorithm to ensure the feasibility of products

Given a set of modules $\mathscr{M}_t$ represented by the vector x_t and a set of product $\mathscr{P}_t$, the problem is to find a feasible bill of material for each product in $\mathscr{P}_t$. The bill of materials must verify that each function occurring in the product is satisfied by exactly one module. A function that is not in the product must not be present in a selected module either. Moreover, the assembly time must respect the maximum duration permitted for this operation (W). To verify this, the following Integer Linear Program, that is a subproblem of the general model given in Section 2, has to be solved for each product $P_i \in \mathscr{P}_t$:

(*feasibility*)

$$\text{s.t.} \quad Ay^i = p_i \tag{6}$$

$$y^i_j \leqslant (x_i)_t \tag{7}$$

$$(y^i)^{\mathrm{T}} w \leqslant W \tag{8}$$

$$y^i \in \{0,1\}^m \tag{9}$$

The feasibility problem is a particular case of the well-known Set Partitioning Problem in which an objective

function has to be minimized. This problem is NP-hard in the strong sense, and so is not tractable for large instances. But, for small instances (fewer than 25 possible functions per product), a standard 0–1 programming algorithm (Wolsey, 1998) gives a true–false answer in less than 10 seconds. Then, the method can be used in an optimization algorithm if it is not often called. The procedure, which will be called `check(input:`$\mathscr{P}_{in}$, $\mathscr{M}_{in}$;`output:`$\mathscr{P}_{out}$) has a set of product $\mathscr{P}_{in}$ and a set of modules $\mathscr{M}_{in}$ as input, and gives, as output, the set $\mathscr{P}_{out}$ containing the products that are not feasible with modules in $\mathscr{M}_{in}$.

4. Computational experiments

This section is organized as follows. First of all, the experimental conditions and instance sets are given in Section 4.1. This subsection also contains the parameters that have been fixed in the method. These parameters were chosen on the basis of preliminary tests. Section 4.2 is devoted to full-problem instances and an extensive study made on them. Finally, Section 4.3 contains results on problems with constraints. For these problems, some constraints have been added to better represent reality. (This paper describes a real industrial problem that concerns the manufacturing of electrical wire harnesses for the automotive industry. Unfortunately, real dataset that describes the product and the manufacturing process cannot be presented.)

4.1. Experimental conditions and instance sets

The tests were carried out on a Pentium 4 (3.00 GHz, 512 Mo RAM). The programs were coded in the Matlab R2007a. Integer Linear Programs are solved by the standard routine `bintprog()`. To select the representatives (`cluster` function), the standard routine `kmeans()` was used.

In order to obtain a fair evaluation of the solution method, the following parameters were fixed, and these parameters were used for all the tests: the number of steps is three with $\beta = 0.3$, 0.2 and 0.1. The maximum number of clusters is 10. The program stops after 5000 simulated annealing iterations or if there is no improvement during 500 iterations in the 3rd step.

Different instances are considered to validate the proposed methodology. We considered full problems and problems with constraints. For each of them, we have the number of functions (q), the minimum (maximum) number of functions q_{min} (q_{max}) that can appear in a finished product, the number of products for which a bill of material has to be computed. A subset of products containing between q_{min} and q_{max} functions is generated leading to the set $\mathscr{P}$. For full problems the set of modules $\mathscr{M}$ is composed of all possible modules.

For large instances and for real applications, it is unrealistic to consider all the possible modules for computational reasons and from a practical point of view. Most of the time, the options follow certain rules which lead to a reduction in the combinatorics of the problem. For example, an option a is in a finished product only if option b is also present; or, conversely, option a is in a finished product only if b is not present. Problems with constraints deal with it, in which case only a subset (10% and 20%, respectively) of all possible modules are available and others are removed. Removed modules in problems with constraints represent modules that cannot be considered for technical or economical reasons. All instances are summarized in Table 1.

In this table, the problem called $F10$ consider the design of $n = 100$ products within a family based on $q = 10$ functions, where each product must contain between $q_{min} = 4$ and $q_{max} = 8$ functions and $m = 1023$ modules are candidate. In problems $C13$-10% and $C13$-20% only a subset of the possible modules (resp. 10% and 20%) is candidate.

In all cases, the time required for final assembly W varies, in order to determine its influence on the bill of materials for the product family. For example, $W = 3$ means that for each product at most three assembly operations are permitted.

4.2. Extensive study on full problems

Full problems are solved 10 times, in order to obtain mean, minimum and maximum values for both the numbers of modules in the solution and computational times.

In full problems, whatever the time of final assembly W, there is at least one solution in the search space (a set of modules) that will permit solution of the problem. Even if $W = 0$, each final product is considered as a module, since no time is available for final assembly. In real-world cases, very small values of W are not considered, as there is no practical reason to do so.

4.2.1. Computational results for problem $F10$

Experimental results for problem $F10$ show that it is possible to assemble any of the 100 different final products with 10 modules if $W \geqslant 8$. If W decreases the number of modules increases. For example, if $W = 6$, around 17 modules are required, while if $W = 4$ solutions vary between 23 and 31 modules (see Fig. 3).

CPU time decreases as W grows, which is what happens to the number of selected modules, and the two dispersion patterns are similar (see Fig. 4).

Large values of W (long final assembly time) allows the company to manufacture all products with a limited number of modules. In fact, if the maximum number of final assembly operations permitted (W) is greater or equal to the maximum number of functions per product

Table 1
Parameters for computational experiments.

Instance name	q	q_{min}	q_{max}	n	m
$F10$	10	4	8	100	1023
$F13$	13	6	10	500	8191
$C13$-10%	13	6	10	500	820
$C13$-20%	13	6	10	500	1640

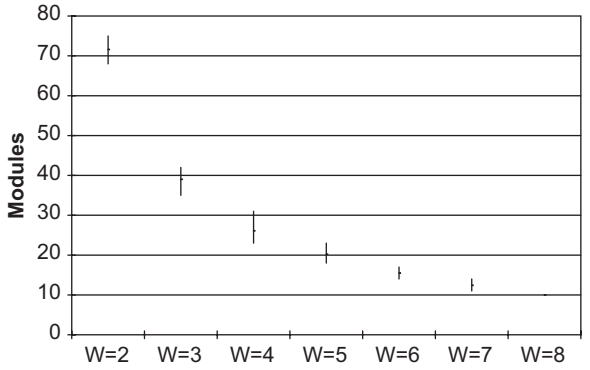

Fig. 3. Problem F10—number of modules.

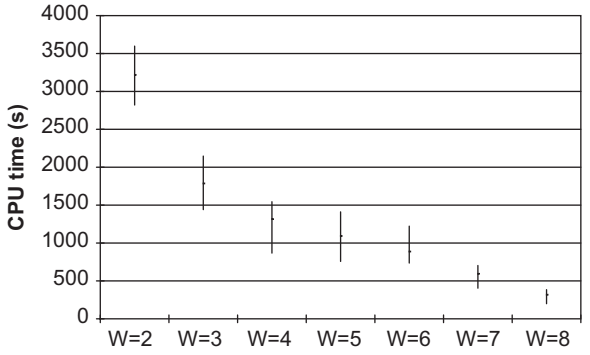

Fig. 4. Problem F10—CPU time.

q_{max}, then the optimal choice is to select all modules that contain only one function. In this case, the optimal number of modules is equal to the number of functions considered in the product family (q). W is so large that it allows the company to manufacture each product from elementary modules.

When W decreases ($W < q_{max}$), it will be necessary to consider supplementary modules that will decrease the time to manufacture final products containing a large number of functions. Then, as W decreases, the number of modules to manufacture increases.

4.2.2. Computational results for problem F13

Problem $F13$ is larger than problem $F10$ in two important ways. First, the number of final products to consider is much larger, which means that a larger product portfolio must be considered in order to satisfy more diverse customer demands, and also the number of functions available for each final product is larger (13 instead of 10), which leads to a larger number of modules to consider for optimization ($m = 2^{13} - 1 = 8191$). In this case, the search space is then much larger.

Results for problem $F13$ show the same evolution as for problem $F10$. When $W \geqslant 10$ (q_{max}), the optimal answer is to select the 13 (q) elementary modules. As the number of modules grows when W decreases (Fig. 5), supplementary modules are necessary to compensate for a shorter final assembly time. Moreover, computational time becomes much greater (almost 10 times as much for the same values of W) (Fig. 6).

4.3. Results on problems with constraints

In order to be able to compare the behavior of problems with constraints to full problems, problems of the same size are considered ($q = 13$, $q_{min} = 6$ and $q_{max} = 10$), but the subset of m modules is limited. Two cases are considered. In the first, $C13$-20%, $m = 1640$ ($\approx 8191 \times 0.2$), while in the second $m = 820$ ($\approx 8191 \times 0.1$). In both cases all modules with only one function are included (13 modules) and the others are randomly selected from among the modules that may be used in at least one final product (no module in m is incompatible with all products in $\mathscr{P}$).

Problem $C13$-20% is presented first because it contains fewer constraints (so that more modules are available and the search space is larger).

Since not all modules are available in the search space, it may be impossible to solve the problem in specific cases. A module may be necessary for the manufacture of a specific product and there may not be one. In this section, we will show results of the optimization process after one and three iterations (plus global cleaning). Some products in $\mathscr{P}$ may not respect constraint (3), so graphics will be provided to reveal the number of products that respect this constraint, called "OK" products, and the number of products that do not respect it, called "NOTOK" products.

4.3.1. Computational results for problem C13-20%

Fig. 7 gives the computational times solely for general information purposes. These values cannot be used for comparison, since the maximum number of general iterations (t_{max} from Fig. 2) may sometimes be a constraint and sometimes not.

The next two figures (Figs. 8 and 9) show the evolution of the number of modules and the number of OK and NOTOK products as a function of W for problem $C13$-20%. Results after one iteration and results after three iterations plus global cleaning are provided.

For $W \geqslant 10$ (q_{max}) with only one iteration (solid line) we obtain the 13 elementary modules and all 500 products respect constraint number (3) (Fig. 8). As W decreases the number of NOTOK products increases (Fig. 9). The first iteration remains efficient for $W \geqslant 8$, but, for smaller values, a considerable number of NOTOK products must still be improved. Moreover, after three iterations (dotted line), we may observe that, even for medium values of W ($W = 6$ and 7), the number of OK products has improved greatly. Even for small to medium values of W ($W = 4$ and 5), results after three iterations are relatively satisfactory, since, with a limited number of modules (between 40 and 60), 70% of the products in $\mathscr{P}$ respect the final assembly time. For really small values of W, it appears that a large number of modules is necessary and that many products may not respect the time constraint.

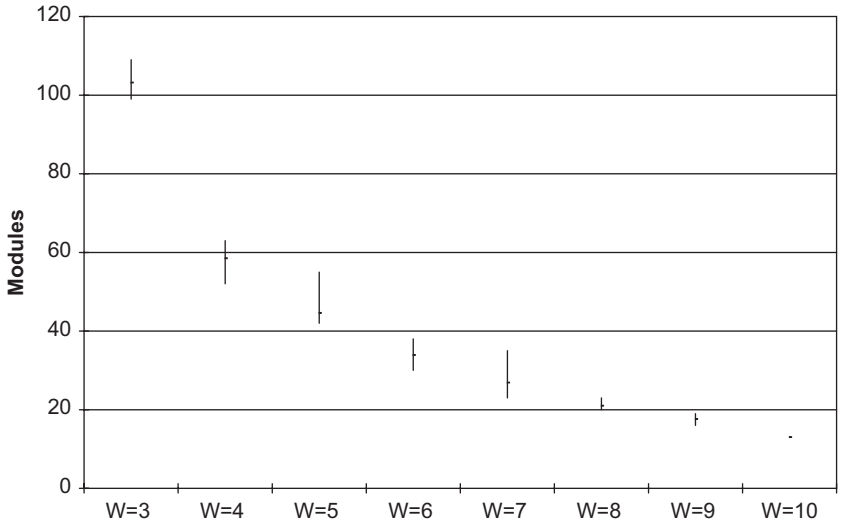

Fig. 5. Problem F13—number of modules.

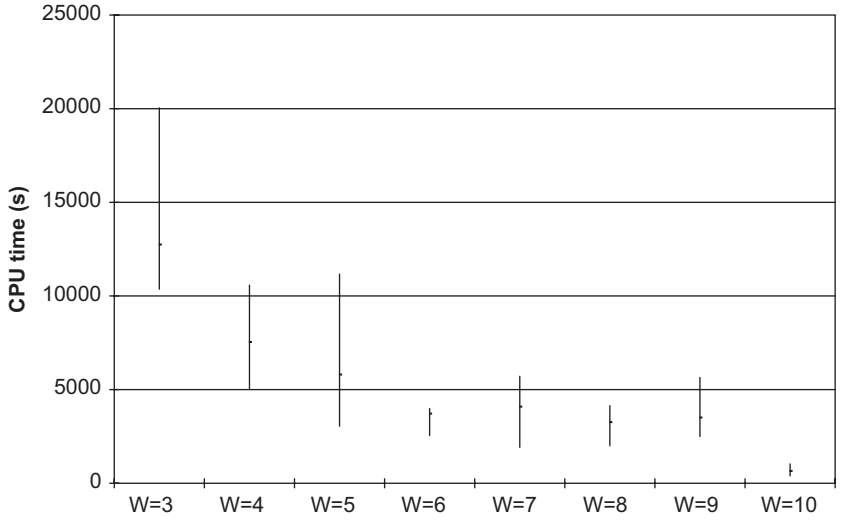

Fig. 6. Problem F13—CPU time.

These results make sense. We saw with full problems that really small values of W will require consideration of almost one module corresponding to each specific product, while in the present problem, many modules are not in the search space, and so many products will not be able to respect the time constraint. For larger values of W, it is possible for more combinations of modules to be assembled in each product in $\mathscr{P}$, and for some of those combinations to still be available with the available modules.

4.3.2. Computational results for C13-10%

As explained previously, Fig. 10 provides an evaluation of the computing time for general information purposes.

In this problem, fewer modules are available than previously (10% instead of 20%); the search space is smaller, but there may be fewer solutions.

The following figures (Figs. 11 and 12) confirm previous conclusions. The first iteration is still efficient for large values of W ($W \geqslant 8$). Three iterations give relatively good results for $W \geqslant 6$, 38 modules permit 70% of the initial products to be manufactured within the allotted time. Besides as fast as $W \leqslant 5$, the quality of the solution drops rapidly, fewer modules are available, and fewer products can be manufactured in the time allotted.

4.3.3. Synthesis on problems with constraints

Sections 4.2 and 4.3 proved that the proposed methodology is efficient for solving product family design

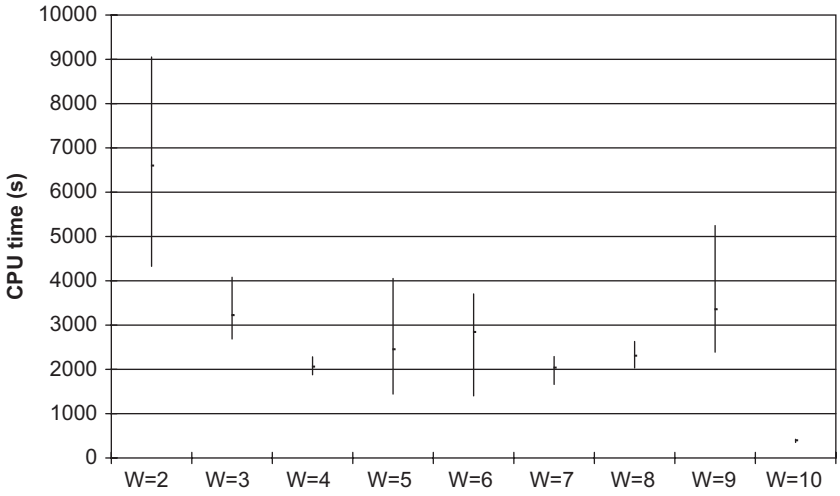

Fig. 7. CPU time for the problem C13-20%.

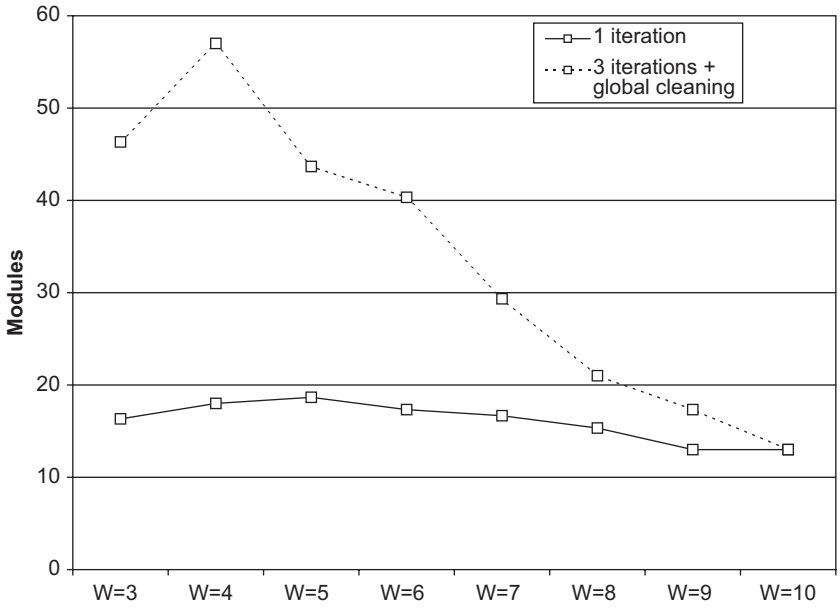

Fig. 8. Problem C13-20%—number of modules.

problems and supports constraints in modules. This section provides an analysis of the impact of constraints on the design problem.

The following figures show the number of modules (Fig. 13) and the number of products (Fig. 14) that respect or do not respect the time constraint (OK and NOTOK products) for the problem with 13 functions depicted and solved in previous sections. The notations are: 100% when the concern is the full problem (100% of the modules are available), which is problem $F13$ in Table 1: 20% for problem $C13$-20%; and 10% for problem $C13$-10%. In all cases, results are provided after one iteration. For problems with constraints results are provided after three iterations, and, for problem $F13$, results are provided after full optimization (final) (all products in $\mathscr{P}$ are then denoted OK).

Figs. 13 and 14 show that when all modules are available (100%), the method always find a feasible solution, whatever the time of final assembly (W), although a different number of iterations may be necessary. In problems with constraints the number of OK products is small for low values of W, but grows with W.

Whatever the constraints, the first iteration leads to similar results in all cases, a similar number of selected modules and a similar number of OK products. After three iterations, with 20% of the modules, a few more modules are selected, compared to the case with 10% of the modules, and also the number of OK products is much

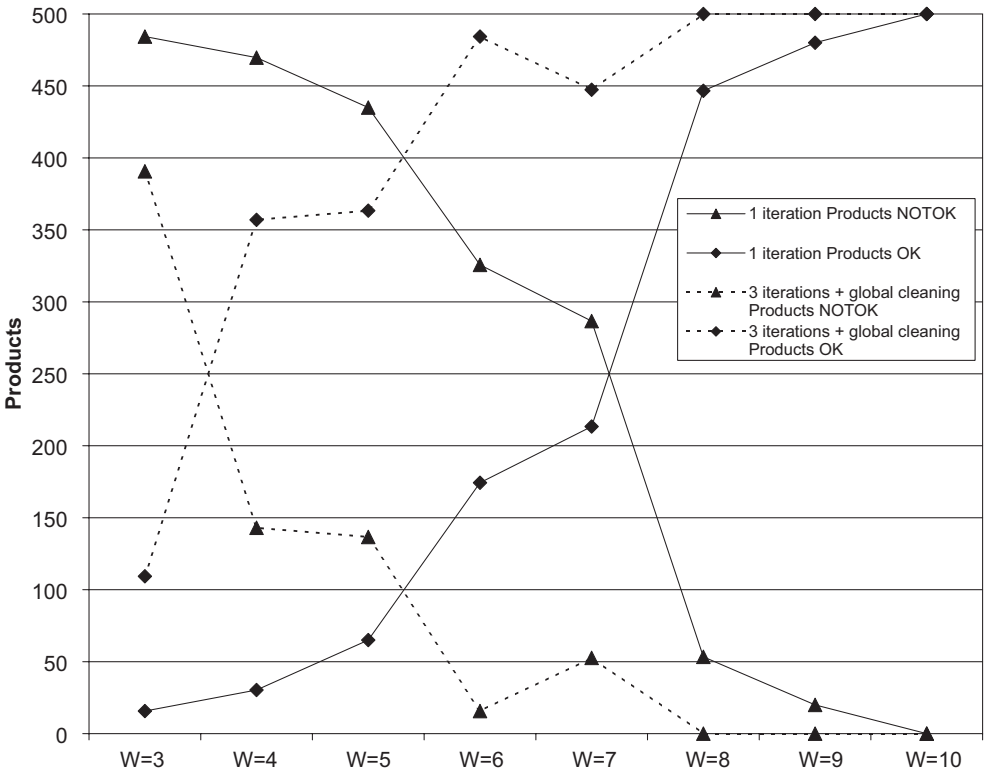

Fig. 9. Problem C13-20%—number of products.

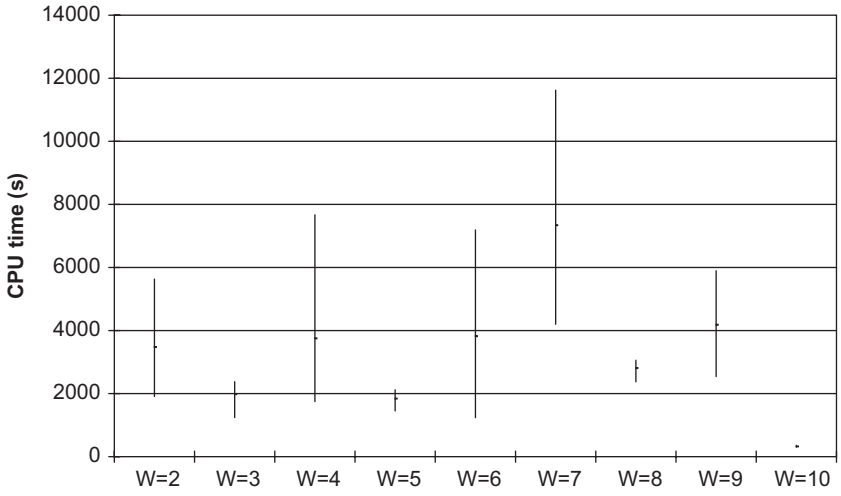

Fig. 10. CPU time for the problem C13-10%.

larger. The method converges to a solution with more feasible products.

5. Conclusion and on going works

This paper has been devoted to a difficult industrial problem arising when companies try to offer a wide variety of products to consumers. In this problem, the choice of modules has to be made efficiently. These modules will be produced for stock, and used in the last stage, which is the assembly line. Several authors considered this problem, using different assumptions—a function can appear twice in a final product, a final product can be substituted by another one containing more functions—but few papers consider the problem in which each final product must correspond exactly to the demand for that product.

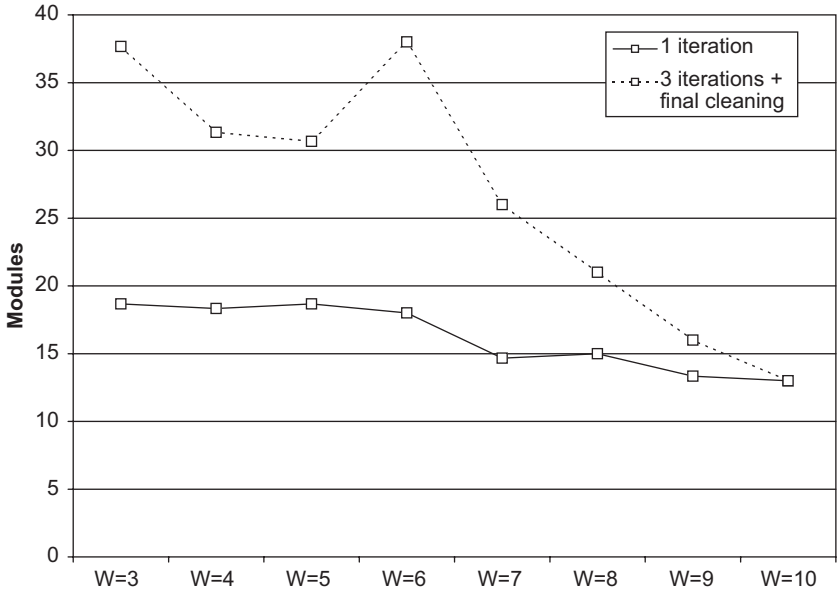

Fig. 11. Problem C13-10%—number of modules.

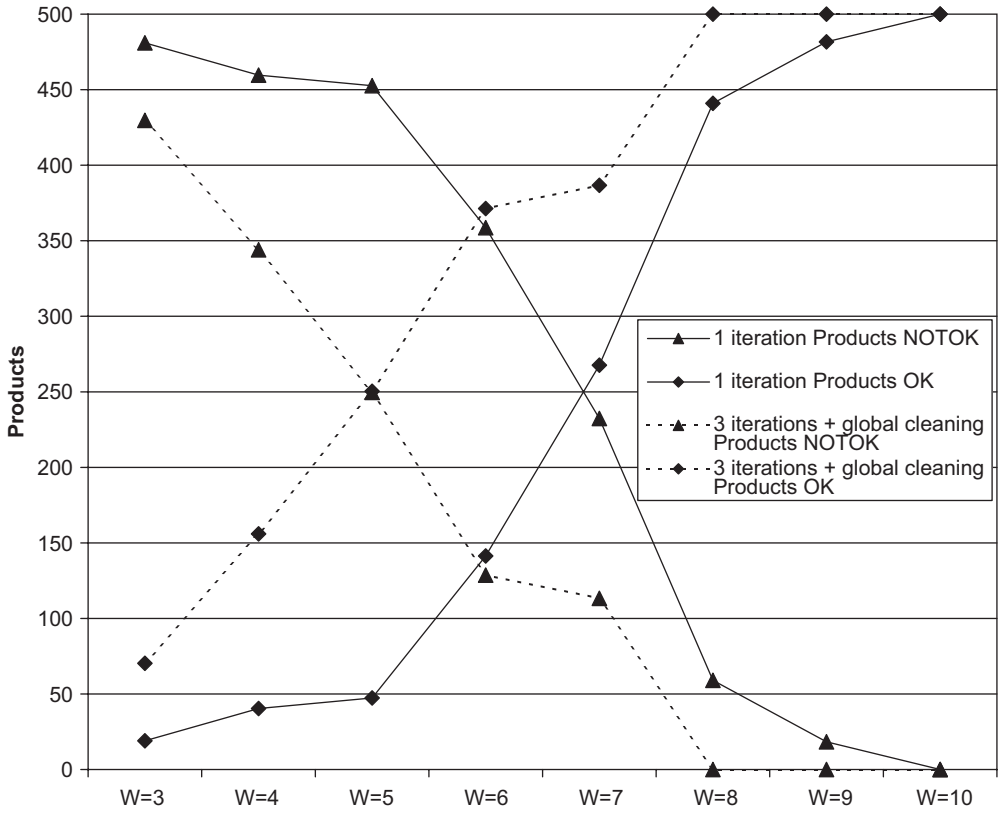

Fig. 12. Problem C13-10%—number of products.

An efficient approximation method has been presented. This method can very efficiently solve a problem with a product having 13 functions. Larger problems may be considered, but more processing time will be required. Our method is almost insensitive to the number of products, and, since representatives are considered, it will

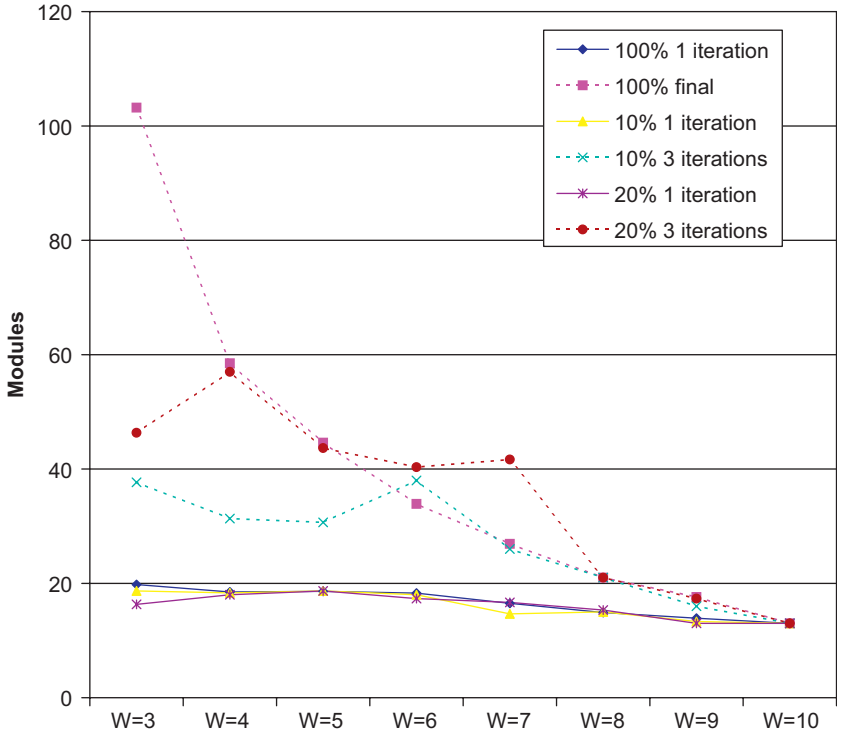

Fig. 13. Synthesis—number of modules.

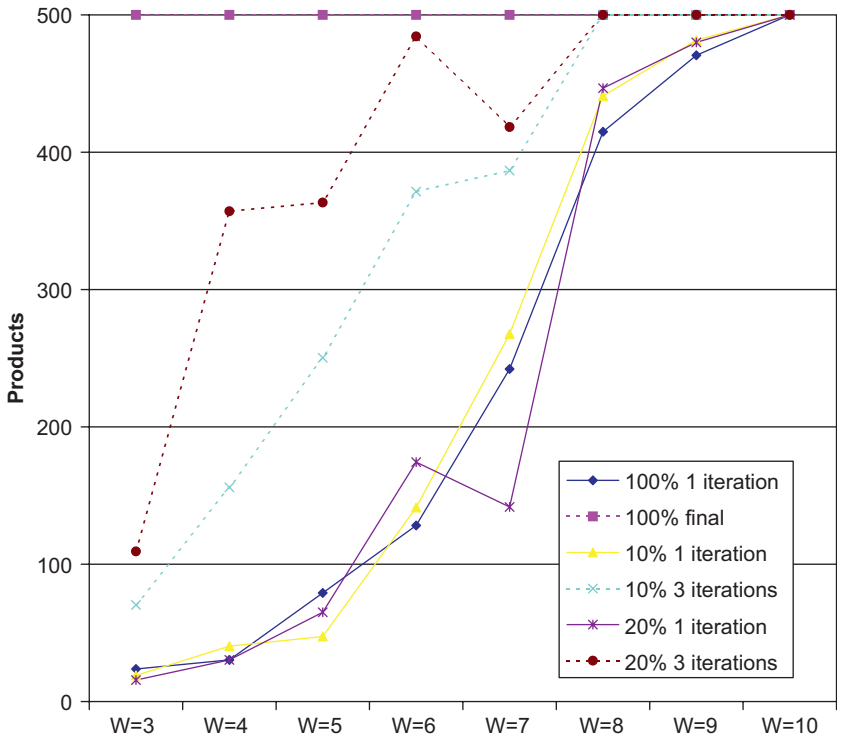

Fig. 14. Synthesis—number of OK products.

change the number of iterations, t, depending of the composition of those products. The result of the method is a list of modules that will permit manufacture of the initial set of products $\mathscr{P}$ with a time constraint.

Our proposed method also supports problems with constraints (e.g. some modules may be unavailable). Constraints may also imply that some products will not respect the time constraint, more constraints may restrict possibilities and fewer products will respect the time constraint.

A limit of this method concerns the modeling of the modules. A large number of available modules (m) implies that a large-scale optimization method will be required to solve it. The problem becomes more difficult if q is also large at the same time. A possible improvement in this area would be to manage a dynamic list of modules, the objective being to consider suppression and insertion procedures for the modules (which may respect the constraints) in order to solve a smaller problem at each step.

Further studies may improve this solution method, which is currently based on a simulated annealing approach, and dedicated heuristics may speed up the process. Another area of improvement could be the way in which the product is modeled; however, it would be challenging to include the assignment of modules to production sites in order to integrate logistical constraints. It could be interesting to consider transportation times in this new model, and the costs incurred, in moving the modules from the production facilities that manufacture them to the facilities that manage the final assembly. The final assembly facility could also be modeled with limited stock capacities, which would limit the number of modules to consider.

In this study, we have focused on the problem of modularization, and we imposed the condition that each product be produced with the exact functions required by the consumer. As we have pointed out, in the current state of the art, another way to approach modularization would be to standardize the products. In other words, replace some products by products containing more functions. Using this strategy, it would be possible to fix the number of finished products to be built. A new strategy might be to find a compromise between these two extreme strategies. The objective function would include the cost of the selected modules and the cost of standardization (replacement with a more sophisticated product).

Yet another way to achieve this would be to consider the expected demand for the products, and consequently the profit: assume a very low demand for a subset of products, but select costly modules. Modeling could then include choosing whether or not to build certain products, and integrating this decision into the optimization objective.

References

Aarts, E., Lenstra, J., 1997. Local Search in Combinatorial Optimization. Wiley, New York, NY, USA.

Agard, B., Kusiak, A., 2004. Data mining for subassembly selection. Journal of Manufacturing Science and Engineering 126 (3), 627–637.

Agard, B., Cheung, B., da Cunha, C., 2006. Selection of a module stock composition using genetic algorithm. In: 12th IFAC Symposium on Information Control Problems in Manufacturing—INCOM 2006, May 17–19, Saint-Etienne, France.

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

da Cunha, C., Agard, B., 2005. Composition of modules' stock using simulated annealing. In: 6th IEEE International Symposium on Assembly and Task Planning—ISATP 2005, July 19–21, Montréal, Canada.

Garey, M., Johnson, D., 1979. Computers and Intractability, A Guide to the Theory of NP-Completeness. W.H. Freeman and Company, New York.

Jiao, J.-R., Simpson, T.-W., Siddique, Z., 2007a. Product family design and platform-based product development: a state-of-the-art review. Journal of Intelligent Manufacturing 18 (1), 5–29.

Jiao, J.-R., Zhang, Y., Wang, Y., 2007b. A generic genetic algorithm for product family design. Journal of Intelligent Manufacturing 18 (2), 233–247.

JoseFlores, A., Tollenaere, M., 2005. Modular and platform methods for product family design: literature review. Journal of Intelligent Manufacturing 16 (3), 371–390.

Kusiak, A., 1999. Engineering Design: Products, Processes, and Systems. Academic Press, San Diego, CA.

Lamothe, J., Hadj-Hamou, K., Aldanondo, M., 2006. An optimization model for selecting a product family and designing its supply chain. European Journal of Operational Research 169 (1), 1030–1047.

Lee, K., Tang, C., 1997. Modeling the costs and benefits of delayed product differentiation. Management Science 43 (1), 40–53.

Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A., Teller, E., 1953. Equation of state calculation by fast computing machine. Journal of Chemical Physics 21 (7), 1087–1091.

Pine II, B., 1993. Mass Customization: The New Frontier in Business Competition. Harvard Business School Press, Boston.

Rao, U., Swaminathan, J., Zhang, J., 2004. Multi-product inventory planning with downward substitution, stochastic demand and setup costs. IIE Transactions 36 (1), 59–71.

Seber, G.A.F., 1984. Multivariate Observations. Wiley, New York.

Simpson, T., Siddique, Z., Jiao, R., 2006. Product Platform and Product Family Design—Methods and Applications. Springer, Berlin.

Spath, H., 1985. Cluster Dissection and Analysis: Theory, FORTRAN Programs, Examples. Halsted Press, New York, 226pp (Translated by J. Goldschmidt).

Swaminathan, J., Tayur, S., 1998. Managing broader product lines through delayed differentiation using vanilla boxes. Management Science 44 (12), 161–172.

Wolsey, L., 1998. Integer Programming. Wiley, New York.