Selecting Optimal Modular Structures using Particle Swarm Optimization

Orlando Durán[1], Luis Perez[2], Nibaldo Rodriguez[1]

[1] Pontificia Universidad Católica de Valparaíso, Chile
[2] Universidad Técnica Federico Santa María, Chile

{orlando.duran, nibaldo.rodriguez}@ucv.cl luis.perez@usm.cl

Abstract. This paper proposes a method which helps users to change their dedicated systems gradually into modular ones. The optimization is achieved through appropriately selecting the subsets of module instances from given sets. The proposed formulation is general in the sense that products can have any number of modules. Particle swarm optimization is used to solve the optimization model. Comparative results are presented using information from exhaustive enumeration.

Keywords: modularity, particle swarm optimization, swarm intelligence, modular design.

1 Introduction

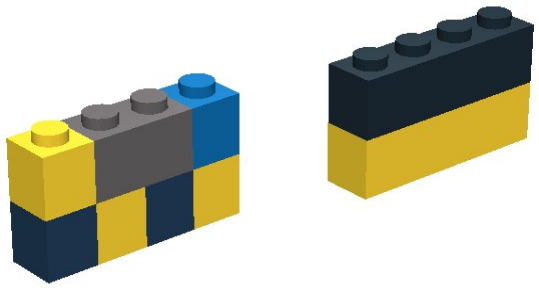
Modularity comprises the use of a finite set of components to meet the infinite changes of the environment; establish the module by reviewing the similarities among the components; keep as much independence of the resulting structures as possible and use different modules for different varieties of assemblies. Modularity is one of the primary means of achieving flexibility, economies of scale, product variety and easier product maintenance and disposal. The main motivation of modularization of a product or a structure is to meet the changing demand for the needs of a product, in particular to achieve rapidly the maximum flexibility at lower costs. The modularization should result in product architectures or structures such that the product can be obtained by

simple assembling pre-existing components. In recent years, the conversion of dedicated structures into modular ones seems to be a trend in the manufacturing field, especially in the search for a greater flexibility. Products built around modular architectures can be more easily varied without adding too much complexity to the manufacturing system, and moreover, a modular architecture makes the standardization of components more possible. At the same time, modularity allows the reuse of tools, equipment and expertise, and avoids costly changeovers for personalized products. Determination of modular configuration is defined in [1] as, "Given a set of candidate modules, produce a design that is composed of a subset of the candidate modules and which satisfies both a set of functional requirements and a set of constraints". From this definition, it can be seen that here we assume that alternative modular configurations for a particular product of structure are established and well known, and modular components and their interactions are predefined and available. There are many domains where a wide variety of modular designs are available. That is, there is more than a unique form to build a given solution or a structure using a set of modular components from a given and a finite set. Depending on the specific domain at hand, many researchers have reported automated systems and methodologies for define one or more modular configurations for a given application, i.e. [2] developed an integrated method for designing modular products. To test and validate the methodology it was applied to a domestic gas detector product family. Hornby et al. [3] developed an automatic design system that produces complex robots by exploiting the principles of regularity, modularity, hierarchy, and reuse. Liu et al. [4] used genetic algorithms for intelligent design of automobile fixtures. [5] developed an automated fixture configuration design system to select automatically modular fixture components for prismatic parts and place them in position with satisfactory assembly relationships. Finally, [6] developed a knowledge-based system for the detailed design of prefabricated building. Despite these clear benefits, a formal theoretical approach for conversion to modular products is still lacking and designers are often skeptical regarding the advantages of modularity. The definition of a modular alternative requires comparative estimations of time, performance and cost among alternative of modular configurations. Recent applications have used cost models and geometric optimizations based on the physical properties (mass, volume) of candidate modules [7]. To evaluate the economic impact of modularity, a cost approach is needed to compare alternative modular cases. The costs differences among modular system alternatives can be used to identify preferred configurations.

This paper proposes a method which helps users to change their dedicated systems into modular ones. The optimization is achieved through appropriately selecting the subsets of module instances from given sets. The proposed formulation is general in the sense that products can have any number of modules. In general, each module may have more than one instance and any each assembled product may have more than assembled combination of modules. The different alternative assemblies may provide the same capabilities and even functionalities that the required ones. Furthermore, current work may accommodate simultaneity constraints where the selection of a particular instance of a module necessitates the use of a particular instance of another module. It is anticipated that the proposed method can be used as a systematic tool in selection of modules instances in designing and assembling modular products or transforming dedicated structures into modular ones.

2 Problem Statement

Consider, as a simple example, a situation where, with a finite number of combinations of modular components, it is possible to construct a body with a given volume. See figure 1, where two alternatives of combinations for constructing the same body are shown. In figure 1(a) seven components were used and in figure 1(b) two components were used. Each modular component has its known cost. If a number of bodies is needed to be assembled at the same moment, and several modules are common to some of these bodies, and, if there is a limited number of each one of the modular components, the challenge here is selecting the optimum combination of modular designs for each one of the lego-like figures (bodies) at a minimum cost. Through modularity, the number of different parts to be purchased for an assembled product set may be significantly reduced while achieving a sufficient variety by combination of different modules.



a) b)

Figure 1. Example of modular alternatives for the same figure.

3 Particle Swarm Optimization

PSO is an evolutionary computation (EC) method inspired by flocking birds ([8]) and has been applied to many different areas. PSO is initialized with a population of random solutions, where this initial population evolves over generations to find optimal solutions. However, in PSO, each particle in population has a velocity, which enables them to fly through the problem space instead of dying or mutating. Therefore, each particle is represented by a position and a velocity. The modification of the position of a particle is performed by using its previous position information and its current velocity. Each particle knows its best position (personal best) so far and the best position achieved in the group (group best) among all personal bests. These principles can be formulated as:

$$v_i^{n+1} = wv_i^n + c_1 r_1^n (p_i^n - x_i^n) + c_2 r_2^n (p_g^n - x_i^n) \quad (1)$$

$$x_i^{k+1} = x_i^k + v_i^{n+1} \quad (2)$$

Where:

- w is inertia weight;
- c_1, c_2 are two positive constants, called cognitive and social parameters, respectively;
- r_1, r_2 are random numbers, uniformly distributed in [0,1];
- n = 1, 2, …, N, denotes the iteration number,
- N is the maximum allowable iteration number.

The first term on the right hand side of Eq. (1) is the previous velocity of the particle, which enables it to fly in the search space. The second and third terms are used to change the velocity of the agent according to *pbest* and *gbest*. Generally speaking, the set of rules that govern PSO are: evaluate, compare, and imitate. The evaluation phase measures how well each particle (candidate solution) solves the problem at hand. The comparison phase identifies the best particles. The imitation phase produces new particles based on some of the best particles previously found. These three phases are repeated until a given stopping criterion is met. The objective is to find the particle that best solves the target problem.

4 Mathematical Formulation

In this paper, we attempt to solve a generalized selection and optimization problem in modular construction of a set of figures. Consider a situation having 'F' figures and `M' modules. Each figure is characterized by its structure, and each structure is comprised for a given combination of modules. Parts may have S alternative structures and all parts are not having equal number of alternative structures (see Figure 2). Under a certain modular configuration for a figure there is associated with a given cost, in correspondence to the total number and types of modules to construct the referred structure or figure. Thus, the problem is to find optimal parts configurations with their modules combinations to minimize total costs. The mathematical model is presented below. As it was mentioned, the problem consists in determining the minimal cost required for assembly all the figures simultaneously, selecting the appropriate combination of alternative assembly for each one of the figures. Here, all the figures are to be assembled simultaneously which it means that the need of some modules could be incremented according to the number of figures that use a specific assembly option that considers the module at hand. Let us introduce the elements of this optimization model. The optimization model is stated as follows. Let:

- M be the number of modules,
- F the number of figures,
- S the number of alternative assemblies or set ups,
- i the index of figures (i = 1,….,F),
- j the index of modules (j =1,…..,M),
- k the index of alternative assemblies (k=1,…..,S),
- A = [a_{ij}] the M x F binary incidence matrix,

Given an incidence matrix P= [p_{ijk}], where:

F_{ijk} = k ,
k= 0,1,…..,K
i = 1,…..,F
j = 1,…….,M
k = 1, …., S

Where *k* represents the number of module j is used by the figure *i* in the alternative of assembly *k*. Each module *j* has a unitary cost c_j. We selected as the objective function to be minimized the cost of a given set of set up of modules to assembly a set of figures to be constructed simultaneously.

$$Min: Z = \sum_{i}^{f}\sum_{j}^{m}\sum_{k}^{s} F_{ijk} C_j A_{ik}$$

$$F_{ijk} = \begin{cases} 1 \text{ if figure } i \text{ uses f modules of the } j \text{ type in the } k \\ \qquad\qquad \text{assembly alternative} \\ 0 \text{ otherwise.} \end{cases}$$

$$A_{ik} = \begin{cases} 1 \text{ if figure } i \text{ will be assembled using the assembly} \\ \quad \text{alternative } k. \\ 0 \text{ otherwise.} \end{cases}$$

Subject to

$$\sum_{k=1}^{C} A_{ik} = 1 \qquad \forall i,$$

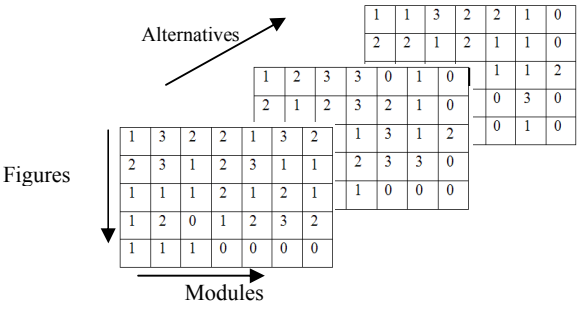


Figure 2. The representation model.

5 Proposed Swarm-based Selection Algorithm

Traditional PSO algorithm was developed for continuous domains. A discrete binary version of the PSO algorithm was developed by [8]. Correa et al. [9] proposed a Discrete PSO (DPSO) algorithm for attribute selection in data mining applications. The algorithm proposed here is based on the DPSO and the concept of proportional likelihoods, also used by [9]. The main difference between the traditional PSO algorithm and the algorithm proposed here is that the proposed algorithm does not use a vector of velocities as the standard PSO algorithm does. It works with a mechanism inspired by the proportional likelihoods concept. According to [9], the notion of proportional likelihood used in the DPSO algorithm and the notion of velocity used in the standard PSO are somewhat similar.

This algorithm deals with discrete variables (assembly alternatives), and its population of candidate solutions contains particles of a given size (n= number of figures). Each component of the particle (vector) takes a value between 1 and k and represents the cell to which the machine is assigned. Potential sets of solutions are represented by a swarm of particles. There are N particles in a swarm. $X(i)$ keeps a record of the best position it has ever attained. This information is stored in a separate particle labeled as $B(i)$. The swarm also keeps a record of the global best position ever attained by any particle in the swarm. This information is also stored in a separated particle labeled G.

The initial population of particles is generated with a series of integer random numbers. These numbers are uniformly generated between 1 and n inclusive. Potential solutions (particles) to the target problem are encoded as fixed length discrete strings, i.e., $X(i) = (x(i;1); x(i;2);.....;x(i;n))$, where $x(i,j) \in 1,....,k$; i =1, 2,...,n and j = 1, 2, ..., p.

For example, given the number of possible or alternative configurations, k = 3 and N = 4, a swarm could look like this:

$X(1) = (1; 2; 1; 2; 3; 3)$

$X(2) = (1; 1; 2; 1; 2; 3)$

$X(3) = (1; 2; 3; 2; 1; 3)$

$X(4) = (1; 2; 2; 3; 1; 3)$

In this example, particle $X(1) = (1; 2; 1; 2; 3; 3)$ represents a candidate solution where product 1 and 3 are configured according the alternative option number 1, products 2 and 4 are configured according alternative 2, and products 5 and 6 are constructed according alternative 3. After the initial population of particles is generated, the process of calculation of the fitness function is performed. In addition, in each iteration, a perturbation subset has been incorporated into the swarm. This perturbation subset is inspired by the bit change mutation approach [10] with which the proposed PSO algorithm can escape from good local optima. The perturbation subset corresponds to the fraction of the total number of particles of the swarm that are mutated when the fitness function tends to premature stabilization. In this work we used 3% of the swarm size. The perturbation subset was selected randomly from the total swarm, and its size is obtained through sensibility tests. The perturbation subset operates as follows:

while (perturbation_element <> perturbation_set_size)

if (x(i,j)= k) then (x(i,j)= p),

p <>k

end while,

where *p* is selected randomly from the possible other alternative configurations.

As commented previously, the algorithm proposed in this paper is based on the notion of proportional likelihoods, and it is based on an adaptation of the concept of velocity proposed by [9] in DPSO. The updating process is based on *X(i)*, *B(i)*, and *G*, and it works as follows. In the context of combinatorial optimization, the velocity of a particle must be understood as an ordered set of transformations that operates on a solution. The transformation of a solution is represented with a term that represents the difference between two positions. Hence, in each case, *(X(i)–B(i))* and *(X(i)-G)* represent the needed movements to change from the position given by the first term to the position given by the second term of each expression. For example, consider the following instances of *X(i)* and *B(i)*:

X(i) = (1; 3; 3; 2; 1; 3)

B(i) = (1; 1; 3; 3; 2; 3)

The difference between *X(i)* and *B(i)* represents the changes that will be needed to move the particle *i* from *X* to *B*. The number μ represents the number of elements different from 0 in the subtraction of *B* from *X*. If the difference between a given element of *X(i)* and *B(i)* is not null, it means that the said position is susceptible to change through operations described below.

A new vector *P* is generated that records the positions where the elements *X(i)* and *B(i)* are not equal. A random number is generated and assigned to β. This number β corresponds to the number of changes that will be made to *X(i)* based on the difference between *X(i)* and *B(i)*; therefore, β is in the interval (0, μ). Then, a set ψ of β randomly generated binary numbers is defined. If the binary number is 1, the change is made; in other hand, if the number is 0, the change is not performed. A similar process is performed to update the particle position in accordance to the best global position (*G*). In case that several of the applied movements involve the same position (machine), the change caused by the global best position, the second operation in our algorithm, has the priority. For instance, if:

X(i) – B(i) = (1-1; 3-1; 3-3; 2-3; 1-2; 3-3) = (0,2,0,-1,-1,0)

The new vector *P(i)* is generated*: P(i)* = (2, 4, 5)

Thus, $\mu = 3$. Suppose that $\beta = 2$ and $\psi = (0, 1, 1)$. That means that positions 4 and 5 will be replaced in *X(i)* by the elements (in the same positions) of *B(i)*, obtaining a modified position vector *X'(i)* shown as follows:

X`(i) = (1; 3; 3; 3; 2; 3)

The process is repeated with the new position *X'(i)* and *G*, obtaining the new position of *X(i+1)*.

6 Numerical Results

The purpose of this section is to show, using a series of numerical examples, how the proposed formulation can be used as an aid to configure a modular system and to provide an assessment of the performance of the implementation of the proposed PSO algorithm. Five test problems were used to evaluate the proposed implementation of the PSO algorithm and for tuning of the perturbation set size. Each experiment considers a problem with 25 figures each one with 2 options of configuration. The module population is comprised by 20 different instances. These five problems were first solved through an extensive search, so that their optimal objective functions are known. Each experiment took about 16 hours on a PC with Intel Pentium 4 running at 3.6 GHz under MS windows. Table I shows the minimum cost obtained in each case.

TABLE I. OPTIMAL RESULTS OF THE TEST PROBLEMS.

Test number	Minimum
1	3583
2	3877
3	2622
4	3426
5	3491

Then the same problems were solved by the proposed PSO algorithm. In the case of the PSO algorithm, nine sets of experiments were run. The tests differ in population size and number of iterations. The parameters used in each one of the PSO algorithm are shown in Table II. All experiments were run 10 times; the results of the first set of experiments are shown in figure 3 and Table III.

TABLE II. PARAMETERS USED IN EACH ONE OF THE PSO ALGORITHM CONFIGURATIONS.

Test number	Iterations	swarm size
1	60	10
2	100	10
3	200	10
4	60	20
5	100	20
6	200	20
7	60	50
8	100	50
9	200	50

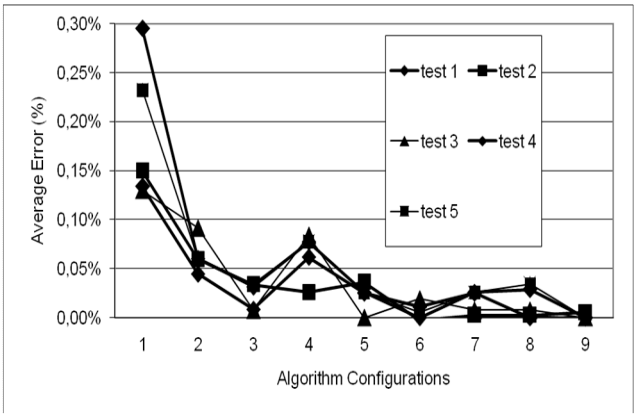


Figure 3. Comparative performance of the PSO algorithm

An observation of the figure 3 and Table III shows that, using the any one of the above detailed configurations of the PSO algorithm parameters it was possible to find the optimal solution at least once. As it was expected, best results were obtained with larger swarms and using more iterations.

TABLE III. RESULTS OF THE NINE PSO ALTERNATIVE CONFIGURATIONS.

Configuration	Avg.Error	Err Std Dev
1	0,19%	0,18%
2	0,06%	0,10%
3	0,02%	0,04%
4	0,07%	0,13%
5	0,02%	0,07%
6	0,01%	0,03%
7	0,02%	0,04%
8	0,01%	0,03%
9	0,00%	0,00%

Table III shows the mean deviations (%) between the optimal solution and the best solution obtained from the 10 executions of the proposed PSO algorithm for each of the five tests. The standard deviation of the error of the different runs is relatively low, indicating that the performance of the algorithm is stable and performs better when the swarm size tend to be larger.

In addition, a set of 12 large problems were randomly generated and solved using the proposed PSO algorithm. These 12 large problems could not be solved optimally. Therefore, it is an open question how far the best solution found by the PSO algorithm is above the optimum value. The details of each one of the 12 test problems are shown in Table IV. In problems like these, exhaustive enumeration would consume CPU time in the order of years. Given the feature of a given test problem, say the problem number 3, the total number of possible combinations corresponds to 10!20 = 1,56*1031. Hence, explicit enumeration is computationally infeasible for this type of problems.

TABLE IV. CONFIGURATIONS OF THE 12 TEST PROBLEMS.

Test	Figures	Options	Modules
1	20	3	15
2	20	5	15
3	20	10	15
4	30	3	15
5	30	5	15
6	30	10	15
7	40	3	15
8	40	5	15
9	40	10	15
10	50	3	15
11	50	5	15
12	50	10	15

The PSO algorithm was applied 100 times to each one of the 12 problems. Computational results are summarized in Table V. Column "Best" shows the total cost of the best solution found by the genetic algorithm in the 100 runs. Column "% |Best-Ave|/Best" shows the deviation between the best solution and the averaged solution over the 100 runs. Column "1% Opt." shows the percentage of solutions that differed by at most 1% from the best solution found. Column "5% Opt." shows the percentage of solutions that differed by at most 5% from the best solution found. The proposed PSO algorithm found solutions that were on average always less than 3,30% above the best solution found. For each problem, 65% on average of the solutions were within 1% of optimally. Finally, almost 100% of the solutions found by the PSO Algorithm were within the 5% of optimally.

TABLE V. COMPARATIVE RESULTS FOR THE TWELVE LARGE PROBLEMS

Test	Best	Average	%	1% Opt	5% Opt
1	1875	1875	0,00	100,00	100,00
2	3632	3633,3	0,04	100,00	100,00
3	4332	4370	0,88	60,00	100,00
4	3822	3825,2	0,08	100,00	100,00
5	4682	4700,6	0,40	100,00	100,00
6	6590	6657,7	1,03	50,00	100,00
7	6514	6535,6	0,33	100,00	100,00
8	6060	6101,5	0,68	70,00	100,00
9	13212	13492,8	2,13	20,00	100,00
10	5167	5203,4	0,70	60,00	100,00
11	9823	9979	1,59	20,00	100,00
12	11589	11971,9	3,30	10,00	80,00

Conclusions

A particle swarm optimization model was presented for selection and optimization for conversion of dedicated structures into modular ones. This problem is found in organizations that face gradual or complete transformation of these products to modular based assemblies. The optimization problem considers the cost involved in using a set of modules to obtain a number of modular assemblies that could substitute a set of non-modular products or structures. A set of problems was generated, and their optimal solutions were obtained with exhaustive search. The PSO model was tested and the results show that the algorithm obtained good solutions in almost all the cases maintaining a low variability of the results. The method is applied to a general problem and is found to be efficient in determining optimum subsets of each module from a given set. The proposed algorithm is computationally feasible for large problems. It finds good solutions with moderate CPU times (assuming that the best solutions found are close to optimum)

References

[1] O'Grady P., Wen-Yau Liang, "An internet-based search formalism for design with modules", Computers and Industrial Engineering, 35(1–2), pp. 13–16, 1998.

[2] Asan U.; Polat S.; Serdar S., An integrated method for designing modular products. Journal of Manufacturing Technology Management, Vol. 15, Number 1 (2004), pp. 29-49(21)

[3] Hornby GS, Lipson H, Pollack JB, Generative representations for the automated design of modular physical robots. IEEE Transactions on Robotics and Automation. Vol.19 pp. 703-719. AUG (2003)

[4] Liu T., Chen CH, Chou JH, Intelligent design of the automobile fixtures with genetic algorithms. International Journal of Innovative Computing Information and Control. Vol.4(10), pp. 2533-2550, OCT (2008).

[5] Babu BS, Valli PM, Kumar AVVA, Rao DN. Automatic modular fixture generation in computer-aided process planning systems. Proceedings of The Institution of Mechanical Engineers Part C-Journal of Mechanical Engineering Science. Vol.219(10), pp. 1147-1152, OCT (2005).

[6] Retik, A., Warszawski, A. (1994) Automated design of prefabricated building, Building and Environment 29 (4), pp. 421-436.

[7] Pandremenos, J., Paralizas, K., Salonitis, K., Chryssolouris G. (2009) Modularity concepts for the automotive industry: A critycal review. CIRP Journal of Manufacturing Science and Technology. 1 (2009) 148-152.

[8] EberhartKennedy, J. & Eberhart, R. C. (1995). Particle Swarm Optimization. In Proceeding of the IEEE International Conference on Neural Networks, Perth, Australia, IEEE Service Center, 12-13.

[9] Correa, E.S., Freitas, A., Johnson, C.G. (2006). A new discrete particle swarm algorithm applied to attribute selection in a bioinformatics data set. In M. K. et al., editor, Proceedings of the Genetic and Evolutionary Computation Conference - GECCO-2006, pages 35–42, Seattle, WA, USA. ACM Press.

[10] Lee,S.,Park, H., M. Jeon (2007). Binary Particle Swarm Optimization with Bit Change Mutation. IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, 90(10) 2253-2256.