Loughborough University
Institutional Repository

Multi-objective optimisation of product modularity

This item was submitted to Loughborough University's Institutional Repository by the/an author.


Metadata Record: https://dspace.lboro.ac.uk/2134/26617

Version: Published

Publisher: Dublin Institute of Technology

Rights: This work is made available according to the conditions of the Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International (CC BY-NC-ND 4.0) licence. Full details of this licence are available at: https://creativecommons.org/licenses/by-nc-nd/4.0/

Please cite the published version.
MULTI-OBJECTIVE OPTIMISATION OF PRODUCT MODULARITY

M. Lee¹, K. Case¹, and R Marshall ²

1. Mechanical and Manufacturing Engineering, Loughborough University, Leics, LE11 3TU, UK
2. Design and Technology Department, Loughborough University, Leics, LE11 3TU, UK

ABSTRACT

The optimal modular configuration of a product’s architecture can lead to many advantages throughout the product lifecycle. Advantages such as: ease of product upgrade, maintenance, repair and disposal, increased product variety and greater product development speed. However, finding an optimal modular configuration is often difficult. Finding a solution will invariably mean trade-offs will have to be made between various lifecycle drivers. One of the main strengths of a computerised optimisation is that trade-off analysis becomes simple and straightforward and hence speeds up the product architecture decision making process. However, there are a lack of computerised methods that can be applied to optimise modularity for multiple lifecycle objectives. To this end, a genetic algorithm based optimisation framework has been developed to optimise modularity from a whole lifecycle perspective, namely, design, production, use and end of life. The paper will look briefly at the optimisation criteria then examine the optimisation framework - in particular the specialised developed genetic algorithm.

KEYWORDS: MODULARITY, GENETIC ALGORITHM

1. INTRODUCTION

Ever decreasing product lifecycles are leading to considerable changes in the way products are being designed. Central to this is the notion of modular product design. The benefits include, shortened design time, improved reliability, reduced construction costs and simplified service and repair. Modular design therefore represents an important means of producing competitive advantages in fast growing and changing markets. Ulrich and Tung [1] define modularity in terms of two characteristics of product design: similarity between the physical and functional architecture of the design and the minimisation of incidental interactions between physical components. There are many more product modularity definitions in the literature. What is generally agreed however is that product modularity is the arrangement of a product’s components into clusters. The clusters contain stronger component interactions and similarities within clusters than between clusters. These interactions and similarities include those which arise from the components’ physical and functional interactions and those which arise from the various processes the components undergo during their lifecycle. The choice of which lifecycle processes to concentrate on as the main drivers for modularity will depend upon the type of product. There have been many previous modular design techniques that have mainly attempted to optimise one modularity objective. Methods that use clustering heuristics have been developed; these techniques only optimise modularity for one objective, for example functional
interactions [2] or testability [3]. Single objective optimisation models have also been developed. Slahieh and Kamrani’s [4] method aims at optimisation of component similarities. Heuristic and non-linear optimisation models have been developed [5] to optimise lifecycle objectives, and manual heuristic based methods have also been developed. Modular Function Deployment (MFD) [6] uses a comprehensive list of modular drivers which can be used to evaluate modules. Stone et al [7] work from a functional basis using energy, signal of material flows between components and use a set of heuristics to form modules. The main problems with the previous methods include: lack of structure; poorly defined modularity evaluation guidelines; no modularity criteria weighting guidelines; Pareto dominance during optimisation; poorly designed optimisation algorithms which can get stuck on local optimums; and the lack of sensitivity analysis. The main contribution of this research is to address these associated problems and create a computerised multi-optimisation framework for product modularity across the whole product lifecycle. This paper looks at the developed optimisation criteria and an optimisation model - in particular the specially developed genetic algorithm.

2. MODULARITY OPTIMISATION FOR THE WHOLE PRODUCT LIFECYCLE

By evaluating modularity from each of the product lifecycle viewpoints, modularity optimisation becomes more organised and logical and this gives rise to several advantages over other methods. Firstly, the organisation of modularity criteria into lifecycle phases creates a clearly defined optimisation problem that can be efficiently handled by the multi-objective algorithm. Next, the importance of each modularity optimisation criteria becomes easier to quantify. Lastly, sensitivity analysis can be carried out. By varying the considered importance of the optimisation criteria the designer is able to study the effects and arrive at the most suitable modular design for the product that is being designed or redesigned. The criteria for modularity optimisation across the stages of the product lifecycle can be seen in figure 1. It is worth noting at this stage that the criteria are by no means exhaustive and a user may wish to include their own criteria in the framework. However one should be careful to ensure that criteria homogeneity is maintained within each phase to ensure optimisation goals are reachable.

<table>
<thead>
<tr>
<th>Module Independence Criteria</th>
<th>Module Coherence Criteria</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design Phase</td>
<td>Localisation of Future Change</td>
</tr>
<tr>
<td></td>
<td>Functional Independence</td>
</tr>
<tr>
<td>Production Phase</td>
<td>Physical Independence</td>
</tr>
<tr>
<td>Use Phase</td>
<td>Physical Independence</td>
</tr>
<tr>
<td>End of Life Phase</td>
<td>Physical Independence</td>
</tr>
<tr>
<td>Make or Outsource</td>
<td>Manufacturing process Similarity</td>
</tr>
<tr>
<td>Design Carryover</td>
<td>Current Product Variety</td>
</tr>
<tr>
<td>Component Life Similarity</td>
<td></td>
</tr>
<tr>
<td>Maintenance and Service Similarity</td>
<td></td>
</tr>
<tr>
<td>Reuse Similarity</td>
<td></td>
</tr>
<tr>
<td>Material Homogeneity</td>
<td></td>
</tr>
<tr>
<td>Recycling Process Homogeneity</td>
<td></td>
</tr>
</tbody>
</table>

Figure 1: Modularity Analysis Criteria for the Whole Product Lifecycle
Within the criteria it can be seen that modularity is measured from the two perspectives of module independence and module coherence. Module independence measures the amount of coupling between modules, and the key to high module independence is to obtain modules that have stronger component couplings within modules and weaker component couplings between modules. Figure 2 illustrates this principle in a design structure matrix (DSM) representation of the product architecture. Module independence is evaluated by using the appropriate evaluation guidelines to evaluate the coupling between all component pairs. The results are then stored in a DSM matrix (e.g. figure 2). Module coherence measures module interactions within modules only. To evaluate the module coherence one must record the similarities between components (according to the given module criteria) in a DSM. Hence components with high module coherence should be placed into the same module during optimisation.

![Figure 2: Clustered DSM example](image)

### 2.1. Modularity Optimisation Criteria for the different Life-cycle Phases

There are three main aims of modularity at the design phase. The first aim is to split a product into modules that can be designed with relative independence from one another, enabling the associated benefits of concurrent design. The second aim is to allow the effects of design change to be isolated within modules. The third aim is to ensure that modules that are to be designed and made or outsourced belong to the same module.

The aim of product modularity at the production phase is to create modules that can be manufactured and assembled as efficiently as possible. For high module independence at the production phase one must aim to minimise the physical interactions between modules. If the physical coupling between modules is too high it may not be possible to manufacture and assemble modules concurrently. Therefore it is highly desirable to group components that have strong physical relationships into the same module and components with weak relationships between modules. Physical couplings between components are evaluated in terms of the geometric attachment and alignment between components. Components that are variants are best separated from components that are common across the product range in order to aid efficient production. Separation of a product into variant and common modules benefits the production phase in a number of ways: firstly, the common modules can be assembled first then variants modules can be added later in the production cycle. This is known as delayed product differentiation, which reduces production lead times. Secondly, production inventory is reduced, as common modules can be assembled and used across a number of product families.

During the product’s use phase a product is likely to undergo repair and maintenance, and so an optimal modular structure can be achieved by grouping components that have similar...

<table>
<thead>
<tr>
<th>Component</th>
<th>Refrigeration Controls</th>
<th>Engine Fan</th>
<th>Radiator</th>
<th>Heater Hoses</th>
<th>Condenser</th>
<th>Accumulator</th>
<th>Evaporator Core</th>
<th>Evaporator Case</th>
<th>Heater Core</th>
<th>Blower Motor</th>
<th>Blower Controls</th>
<th>Air Controls</th>
</tr>
</thead>
<tbody>
<tr>
<td>Refrigeration Controls</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Engine Fan</td>
<td>0</td>
<td>9</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Radiator</td>
<td>0</td>
<td>9</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Heater Hoses</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Condenser</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Accumulator</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Evaporator Core</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Evaporator Case</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Heater Core</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Blower Motor</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Blower Controls</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Air Controls</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 2: Clustered DSM example
maintenance and service needs and similar component lives into the same module. At the same time each module must remain as independent as possible, with minimal physical interaction between modules. This ensures that worn out modules can be replaced or repaired efficiently.

Creating optimal modularity at the end of life phase requires that components are grouped according to their reusability and recyclability. However, at the same time, the physical coupling between modules must remain low to ensure minimum disassembly effort.

3. SOFTWARE OPTIMISATION – MODULE FORMATION

The goal of the GA-based optimiser is to find the optimal modular structure from within a number of matrices (representing component interactions). This is achieved by adjusting module size and membership until product modularity is optimised.

3.1 Specialised Grouping Genetic Algorithm

Many researchers have concluded that GA-based methods are promising for optimisation problems. GAs use the Darwinian theory of evolution to solve a range of optimisation problems. The basic idea is to code the optimisation problem into chromosomes and selectively mate promising chromosomes (exchanging genetic material) in an interactive process until an optimal solution evolves. To do this, a GA needs a well-defined crossover operator (to exchange genetic material) and a mutation operation to introduce new genetic material and stop the solution space converging too quickly. The modular design optimisation problem can be described as a grouping problem, as we are trying to group a predefined number of components into a given number of modules in a manner that will maximise modularity according to the defined fitness function. Traditional GA’s have the inability to preserve the integrity of groups and pass on useful genes to the next generation. Therefore a specially modified version of Falkenauer’s [8] Grouping Genetic Algorithm (GGA) is used for the modular design problem. The important point is that the genetic operators will work with the group part of the chromosomes.

3.2. Chromosome Encoding

The chromosome encoding scheme is simple but gives the required level of information necessary to perform the specialised GGA. The encoding of the GA chromosome is basically an array of real numbers. These array numbers represent module number membership, their positions within the array, corresponding to a component position in the matrix. For example in the Chromosome (figure 3) array element 1 corresponds to component 1, the value of array element 1 is 5, meaning that component 1 belongs to module 5.

![Figure 3: Chromosome encoding](image-url)
3.3. Initial Population Generation

Creating a good initial population of chromosome arrays is critical in ensuring that the GA converges within a reasonable time frame. It is also important the population is diverse enough to ensure that a good variety of genetic material is available for crossover. This will help to avoid problems such as premature convergence or converging on a local rather than a global optimum. The initial population is generated as follows:

Step 1. Select a component with a high number of component interactions and allocate it a module number.
Step 2. Select a component that has a low level of interaction between other components already allocated to a module and allocate the component a module number.
Step 3. Repeat step 1 until each module has one component allocated to it.
Step 4. Randomly select a component and allocate it to the module that gives the greatest improvement in modularity fitness.
Step 5. Repeat step 1 until all components have been allocated to the modules.

Step 1 is performed by looping through all the components and recording the highest interaction value. The highest value is then multiplied by a random value between 0.7 and 1.0 and components are then randomly selected until this value is reached. A similar process is repeated for step 2. This process ensures that good module seed points are obtained during initial population generation. The element of randomness ensures that the population will not contain the same module seeds, which would create a similar module structure for each chromosome.

3.4. Crossover Operator

The crossover operator in a GGA works with the group’s part of the chromosome, in this case the module groups. This ensures that the genetic information from good module groups is carried over to the next generation. The crossover operation takes place as follows:

Step 1. Select two parent chromosomes, mum and dad.
Step 2. Select half of mum’s chromosome content by random selection of module groups within the array e.g. module 1,3,6.
Step 3. Select half of dad’s chromosome content by random selection of module groups within the array e.g. module 2,4,5.
Step 4. Create a new offspring chromosome by inserting mum’s selected contents into an array and then insert dad’s contents. If the array elements are already populated with mum’s content, then overwrite with dad’s.
Step 5. Repopulate any empty array element in the new offspring with new content using step 4 of the initial population generation method. If the number of modules within the new offspring is less than required number, then first perform step 2-3 of initial population generation.
Step 6. Create offspring 2 by repeating steps 2-5, only swapping mum with dad.

3.5. Mutation Operator

The mutation operator for a GGA also works with the group’s part of the chromosome. For the MGGA the mutation operation is performed using steps 1-4 below. Mutation will always occur straight after a crossover, on a newly generated offspring chromosome.

Step 1. Randomly generate a number between 0.0 and 1.0. If the number is less than the mutation rate, then perform mutation.
Step 2. Randomly select a module group from the offspring chromosome – only components in this group will be affected by the mutation.
Step 3. Randomly select a component from the selected module group and allocate it to the module group that gives the greatest improvement in modularity fitness.

Step 4. Repeat step 3 until all components in the selected module group have been allocated to modules.

For the MMGA the mutation rate changes as the algorithm progresses. Initially the mutation rate will be low, slowly becoming higher as the population starts to converge. The logic follows, that at the beginning of the GA run there will be a large variety of genetic material in the population, but as the algorithm progresses and begins to converge to the optimum, the genetic material in the population will be less diverse, so at this point the mutation rate is increased which will introduce a greater amount of genetic diversity when it is most needed.

3.6. Goal Programming

Attempting to maximise one lifecycle phase’s modularity will often mean a reduction in another phase’s modularity. If the optimisation technique is not properly designed this can lead to a Pareto dominated solution (a solution that is dominated by one or more objectives). To address this problem goal programming is used. The goal programming technique provides a balanced and controllable optimisation. An outline of the application of the goal programming and the GA based optimisation method is as follows:

Step 1. Set the required number of modules.

Step 2. Optimise modularity for each lifecycle phase:

2.1 For lifecycle phase n, run the GA with module independence coherence and module goals set equal

2.2 Examine the module grouping results and if satisfactory then go to step 2.5

2.3 Adjust the goal deviation weights for moderate independence and module coherence and rerun the GA

2.4 Examine the module grouping results and if satisfactory then go to step 2.5

2.5 Repeat steps 2.1 to 2.5 for lifecycle phase n

Step 3. Use the optimal modularity results from step 2 and set the goal maximums for each lifecycle phase

Step 4. Run the GA to minimise goal deviations from the goal maximums

Step 5. Perform sensitivity analysis by adjusting the goal deviation weights for the four lifecycle phases and repeat step 4

Step 6. Examine the module grouping results and if a satisfactory solution exists then end the process

Step 7. Adjust the required number of modules and go to step 2

As seen in step 2 the method first performs a GA optimisation for each phase of the lifecycle, where the goal is to find the optimal balance between the corresponding module independence and coherence criteria by using equations 1 and 2 (below). Once this has been performed maximum fitness scores for each lifecycle phase will be known and the GA is then run to optimise modularity for the whole lifecycle. The maximum fitness scores for each phase will effectively become the goal maximums. The goal is then to minimise all deviations from each goal's maximum using equation 3 (below). By adjusting goal weights it is easy to perform a sensitivity analysis of various modular architecture alternatives to enable the designer to consider the merits and trade-offs between different solutions.

\[ MC = \frac{CI_{\text{max}}}{\sum_{i} CI_{\text{internal}}} \]

\[ Lcp = \text{lifecycle phase}, \ m = \text{module number} \ CI_{\text{max}} = \text{max number of module coherence interactions} \]
\[ C_{\text{internal}} = \text{actual number of module coherence interactions} \]

\[ MI = \frac{1}{\sum_{m} I_{\text{total}}} \]

\[ L_{\text{cp}} = \text{lifecycle phase}, m = \text{module number} \]

\[ I_{\text{total}} = \text{total number of module independence interactions} \]

\[ C_{\text{internal}} = \text{actual number of module coherence interactions} \]

\[ I_{\text{total}} = \text{actual number of module independence interactions} \]

\[ T M = \min((G_{d_{\text{design}}} \times w), (G_{d_{\text{production}}} \times w), (G_{d_{\text{use}}} \times w), (G_{d_{\text{eol}}} \times w)) \]

\[ G_{d} = \text{deviation from goal and } w = \text{goal deviation weight} \]

4. **EXAMPLE CASE STUDY – CAR CLIMATE CONTROL SYSTEM**

The car climate control system has been used in various studies, so makes an ideal case for comparison purposes. The aim of the case study is merely to demonstrate the potential of the method as a means of optimising multiple modularity objectives. Therefore the modular criteria scores entered into the software are by no means completely accurate and are based on the

![Modular architecture of car climate control system with equally weighted goal deviation for each life cycle phase](image_url)
authors’ best judgements so will need to be quantified by further research. However the functional and physical interactions were based on previous work [2]. Example results of the software optimisation can be seen in figure 4.

Figure 4 shows the modularisation of the product with the lifecycle deviation goal weightings set equally. By changing the goal deviation weightings of the four lifecycle phase’s sensitivity analysis was performed - partial results of which can be seen in figure 5. From these sensitivity plots the design team would be able to arrive at an optimum modular design solution. A chosen modular solution can be further improved by changing component attributes. This has been examined as part of this research but is out of the scope of this paper and has been excluded.

<table>
<thead>
<tr>
<th>Design Goal Deviation weight</th>
<th>Goal Deviation</th>
<th>Production Goal Deviation weight</th>
<th>Goal Deviation</th>
<th>Use Goal Deviation weight</th>
<th>Goal Deviation</th>
<th>End of Life Goal Deviation weight</th>
<th>Goal Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>50%</td>
<td>42%</td>
<td>100%</td>
<td>22%</td>
<td>100%</td>
<td>21%</td>
<td>100%</td>
<td>23%</td>
</tr>
<tr>
<td>100%</td>
<td>25%</td>
<td>100%</td>
<td>28%</td>
<td>100%</td>
<td>29%</td>
<td>100%</td>
<td>26%</td>
</tr>
<tr>
<td>150%</td>
<td>20%</td>
<td>100%</td>
<td>35%</td>
<td>100%</td>
<td>34%</td>
<td>100%</td>
<td>37%</td>
</tr>
<tr>
<td>200%</td>
<td>16%</td>
<td>100%</td>
<td>38%</td>
<td>100%</td>
<td>38%</td>
<td>100%</td>
<td>38%</td>
</tr>
</tbody>
</table>

Figure 5: Results of a sensitivity analysis, adjusting the design goal deviation weight

5. CONCLUSIONS AND FURTHER WORK

It has been seen that optimisation of a product’s modularity is a desirable but often complex task. However using the proposed computerised methodology, modularity optimisation is a less laborious and time intensive task, making it more approachable for the designer or organisation to consider. Future work will focus on further assessment and refinement of the technique.

6. REFERENCES