ID: 164 (Draft, Accepted for publication)

A Rapid Algorithm for Multi-objective Pareto Optimization of Modular Architecture

Roozbeh Sanaei[1], Kevin Otto[2], Kristin Wood[1], Katja Hölttä-Otto[2]

[1]SUTD-MIT International Design Centre, Singapore University of Technology and Design, Singapore;
[2]Department of Mechanical Engineering, Aalto University, Finland

ICED17: 21st International Conference on Engineering Design
University of British Columbia, Vancouver, Canada
Aug. 21-25, 2017

Full Paper Submission - DesMet
Topics: Design methods and tools
Keywords: Computational design methods, Product architecture, Optimisation

Abstract
Assigning components and functions to modules early can facilitate faster and higher quality results in the latter stages of development and manufacturing. Methods have been developed to suggest suitable architectures using a variety of metrics that measure the ideality of the modularity. However, different modularity criteria considered are often in conflict with each other and improving one is not achievable without a compromising effect on another. To investigate this, we explore using multiple metrics such as defined in the modular function deployment and consider modularization as a multi-objective optimization problem. We develop here a new multi-objective search algorithm that is able to quickly find non-dominated Pareto-optimal architectures. The algorithm is demonstrated for a cordless vacuum cleaner. We further compare the performance of the algorithm with previous work using the IGTA+ algorithm for multiple-objective clustering. The results show the new algorithm improved the computation time of generating the Pareto-optimal surface by more than two orders of magnitude, reducing the 54 component vacuum cleaner modularization search from 192 hours to 24 minutes.

1 INTRODUCTION

Modular systems are composed of subsystems that perform distinctive functions independently (Gershenson et al., 2003; Otto and Wood, 2001). Modular architectures can help developing resource-efficient products in variety of ways. The definition of production modules can have a large impact on the overall manufacturing cost (Rogers and Bottaci, 1997). Modular platform-based product family design allows a family of products with distinguishable overall functions and customer perceptions to be developed using a small set of modules designed before-hand (Du et al., 2001; Gershenson et al., 2003; Hölttä and Otto, 2005; Stone et al., 1998). Modular products also allow for designing products that can be updated over time or easily repaired when damaged through module replacement (Dahmus et al., 2001; Salonen et al., 2008). A product modularized considering recyclability can facilitate recycling (Gu and Sosale, 1999). Modules from older product generations can be reutilized in developing new products (Kimura et al., 2001). All of these benefits has therefore drawn the attention of a wide-range of industries to the modularization approach (Ulrich, 1994).

In pursuing a suitable product architecture modularization, various metrics have been developed to measure the modularity, and various procedures have been proposed to search for more 'ideal' modular architectures. One approach is manually implementing function based heuristics which were derived from patterns observed in successful products (McAdams et al., 1999; Stone et al., 2000, 1998, 1998; Zamirowski and Otto, 1999). These heuristics provide alternative suggestions and do not posit any uniqueness for a grouping of elements (Holtta and Salonen, 2003). This methodology works well for many products and systems but with larger-scale systems, the number of possible combinations increases and may leave too many options for a designer. For large-scale systems, finding a modularization choice using these heuristics may become non-trivial.

An alternative approach to function structure modelling is to use optimization-algorithms that assess a large numbers of architectures probing for architectures with optimal value of a selected modularity metric (Sharman and Yassine, 2007; Yu et al., 2003). Finding an 'ideal' architecture that considers all relevant objectives is generally not feasible as objectives often trade-off and improving one is not possible without worsening another objective (Sanaei et al., 2015). Accordingly, instead of searching for an ideal architecture, we consider here generating Pareto-frontiers consisting of non-dominated Pareto-optimal architectures. Pareto-Optimal or non-dominated solutions are optimized such that none of their objectives can be improved without compromising effect on another objective. However, common modularity approaches using optimization algorithms can be prohibitively slow to be practiced interactively, and further computationally the final result can have inaccuracies if the Pareto-frontier is not convex. Here, we propose and demonstrate an algorithm which is considerably more efficient than common algorithms available at determining the Pareto frontier of modular architectures, and also does not presumes convexity of the Pareto frontier. Considerable improvement in the computational efficiency of the algorithm allows for interactive use, for comparisons and interactive changes to the architecture.

The subsequent sections of this paper are structured as follows. First, we discuss the related literature on which this research is based. Then the design principles and the procedure of a proposed new clustering algorithm is discussed. Afterward, the approach is explored on a high granularity DSM from a vacuum cleaner. The results is compared with previous works using the vacuum cleaner example.

2 RELATED WORK

Considering product architecture as the configuration, blueprint, or outline by which functional elements are mapped into physical subsystems, an architectural element may be a single component, a group of components, or a sub-system (Eppinger and Browning, 2012; Otto and Wood, 2001). For our purposes, modules are the grouping of elements in a product architecture, each with a well-defined interface(Van Wie et al., 2001). Design structure matrices (DSM) and graphical function structure diagrams are two common approaches for capturing and representing product architecture.

In the graphical representation, the high-level function is decomposed into sub-functions and the architecture depicted as a labelled graph where the nodes and edges represent sub-functions and their interactions (Stone et al., 1998; Stone and Wood, 2000; Zamirowski and Otto, 1999). These heuristics do not force fixed axioms for clustering product architectures but provide suggestions on different alternatives for chunking. Three types of flow interactions are commonly considered amongst elements: energy, material, and information flows. In their study of a jet engine, Sosa et al. (2003) expand this to four generic interaction types to include 'structural' as that captures mechanical forces, rather than considering structural forces as a force energy flow.

Another line of research has investigated metrics to assess a product architecture, to help guide a designer in partitioning. Erixon and Ericsson (1999) have developed modular function deployment (MFD) and defined several modularity drivers, to assess an architecture. With MFD, the intent is to find strategic similarities between components and group them into modules. A special matrix designated the *module indication matrix* (MIM) captures the strategic intent of each technical solution or a component. MFD introduces a set of modularity drivers to assess quality of modularization in different aspects. Modularity drivers are selected based on the product-related company-specific strategic requirements which implies there is no single 'ideal' architecture. Hölttä and Otto (2003) developed similar scores based on earlier heuristics developed by Rechtin (1991) and others. A broader list of 23 modularity drivers were introduced by Blackenfelt (2001). The metrics all relate to strategic issues to be considered at the architectural phase. While these metrics cover the range of considerations typically needed, they also therefore typically conflict. Making a low cost architecture can drive a designer towards integration of components, making a flexible architecture can drive a designer towards separation of components. We make use of MFD metrics in this work, as complementary to the modularity heuristic work. Here we assign a score to each relevant modularity driver and take them as objectives of our modularization.

DSMs are the adjacency matrix of a functional diagram and encompass the interactions among components and the type of interaction (e.g., energy, material, information, spatial). DSMs provides a systematic representation that can be easily manipulated with software (Eppinger and Browning, 2012). On the other hand, function structure diagrams may be easier for manual graphical use as functional graph models (Sanchez and Mahoney, 1996).

Guo and Gershenson (2003) reviewed eight modularity measures in pursuance of a modularity measure that can be used effectively to guide product redesigns. They introduced a new measure *modality* that is independent of DSM size; this measure has positive dependence on interactions within modules and negative dependence on interactions among them. Hölttä-Otto built on this work by dividing modularity metrics into two main classes (Hölttä-Otto et al., 2012), connectivity-based metrics and similarity based metrics. The connectivity class constitutes the majority of metrics developed to date and is concerned with the coupling of components within and amongst modules. All connectivity metrics consist of a combination of two main measures in various ways, the first is the level of independence of a module from the rest of the system and the second is the integrality of a module within themselves (Hölttä-Otto et al., 2012). The similarity based class of metrics measures the degree of similarity of elements within modules, this could be similarity in materials, manufacturing processes, suppliers, functions, life-cycle, etc. Hölttä-Otto observed that few metrics attempt to combine these measures to increase independence of modules along with degree of similarity elements within modules. We consider this problem here, and allow for trade-off of multiple metrics.

There are also metrics intended to evaluate individual modules rather than an entire architecture, such as the *Module Strength Indicator* developed by (Van Beek et al., 2010). This metric is formulated to measure the internal connectivity within a module and the external connectivity of the module. They used MSI as an objective function for optimising architectures utilizing a k-means clustering algorithm. This algorithm can find a local optimum quickly.

Not all proposed modularity metrics can guide clustering algorithms, two such examples are *Singular Value Modularity Index* (SMI) and the and the *Non-Zero Fraction* (NZF) metric, both introduced by

Hölttä-Otto and de Weck (Hölttä-Otto and De Weck, 2007). The SMI captures the degree of integrality of components within a product architecture, and the NZF assesses the sparsity of connections among components. In contrast, the *Minimum Description Length* (MDL) (Yu et al., 2007) is information theoretic based metric that measures the amount of information needed to describe module size and connectivity within each module and among modules. While it has been shown to be effective in driving module clustering algorithms, it is also dependent on the size of the system, larger systems in general have better MDL.

Fernandez (Gutierrez, 1998) and Thebeau (Thebeau, 2001) introduced two separate modularity metrics, one for measuring integrity within modules, *intra-cluster-cost*, and a second to measure the independence of modules called *extra-cluster-cost*. Thebeau also introduced *ClusterBid* as a degree of proximity between a component and a module. Lower values of intra-cluster-cost and extra-cluster-cost equate to higher levels of integrity and independence. Intra-cluster-cost and extra-cluster-cost are shown to be in conflict with each other (Sanaei et al., 2015). It is also shown that extra-cluster-cost has positive dependence on the number of modules while intra-cluster-cost has dependence on variance of module sizes (Sanai et al., 2016).

The *IGTA* algorithm (Idicula-Gutierrez-Thebeau Algorithm), cited as one of the most common algorithms used for product modularization to date (Borjesson and Hölttä-Otto, 2012), is a stochastic hill climbing algorithm built upon *total-cost*, the weighted sum of intra- and extra-cluster-cost. Borjesson and Hölttä-Otto (2012) improved the IGTA clustering algorithm to obtain useful results faster, *IGTA+*, which provides the fastest published algorithm for clustering DSMs. The IGTA+ clustering algorithm has been integrated with module drivers from MFD (Borjesson and Hölttä-Otto, 2014) to consider strategic similarities between components. In previous work (Sanaei et al., 2015) we framed IGTA+ to perform multi-objective optimization and Pareto Frontier generation by optimizing a weighted summation of multiple objectives using IGTA+. In this paper we introduce a new algorithm that out-performs IGTA+ computationally in generating a Pareto-optimal surface of solutions by several orders of magnitude. Further, to benefit from multi-cores available on most computing systems, we adapt the algorithm to run in parallel on multi-CPUs.

3 ALGORITHM: DESIGN PRINCIPLES AND PSEUDO-CODE

Given the large search space, DSM optimizations typically make use of heuristic optimization methods. The common sequence in heuristic optimization algorithms is outlined in (Osman and Kelly, 1996). A series of solutions are generated and their quality is assessed according to objective functions. Iteratively new solutions are generated through combination and modification and filtered before proceeding to a next iteration. The space of solutions is generally astronomically large. One common strategy is finding ways to generate solutions more likely close to optimal. What puts a solution in close-to-optimal region varies problem by problem and there is no general-purpose optimization approach (Wolpert and Macready, 1997).

Our algorithm consists of three distinct stages. First, we find all possible breaks in connections that thereby bisect the architecture into two halves. We refer to these as *cuts*. Architectures with more than two modules then can be generated through combinations of these cuts. Second, we increase the architectural space coverage by adding new cuts or exchanging and replacing cuts among modularizations in an evolutionary approach. Third, we explore the neighbourhoods of obtained architectures by moving components between modules to further refine the search.

There are two novel factors in this approach that contribute to computational efficiency for Pareto frontier identification. First, generating modularizations by overlapping of bi-sectioning cuts is more efficient than random generation. Second, optimizing space coverage rather than metric minimization is more efficient at generating the Pareto Frontier. In combination, we find these two factors increase the speed of Pareto frontier generation by several orders of magnitude, from hours down to minutes for typical problems of dozens of elements. In the following, we described each of these in more details.

3.1 Generating architectures

A common representation of modules in product architectures is the set of elements that each module includes. Alternatively, modules can be identified through the set of links that module boundaries cut through. These two different representations provide different methods to generate new architectures. The first representation generates new architectures by moving components from one module to another. The second representation generates new architectures by replacing and adding new cuts. The difference between these two representations is schematically illustrated in Figure 1. The first approach is that traditionally taken in DSM heuristic search such as IGTA. The second approach is novel, and explored here.

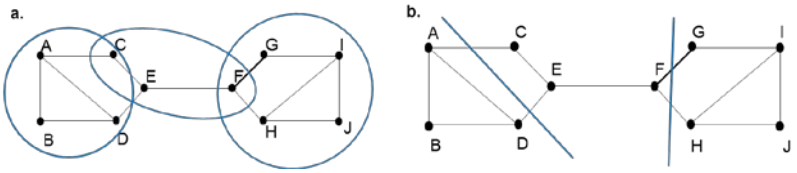

Figure 1.a Illustration representing modularity scheme by sets of components each module contains, Figure 1.b Demonstration of modularity scheme by their interfaces.

As illustrated in Figure 1, there are limited sets of cuts that can bisect an architecture into two halves. For the function structure diagram shown in Figure 1 there are 43 distinct cuts that can bisect the diagram into two parts. To find these cuts, we first find the individual edges that when removing them would bisect the graph. For the rest of the edges, we remove a second edge, and repeat until finding all possible edges that, when removed, bisect the graph. To then cluster the diagram into more than two halves, modules can be generated by overlapping multiple bisecting cuts, as illustrated in Figure 2. Since the number of cuts is smaller than number of modules and number of links within the set of cuts is the number of links crossed by the module interfaces, we can bound the number of modules and links that cut interfaces. This prevent exploration of costly architectures that have large numbers of extra-cluster links. Therefore, generating modules by combining bi-sectioning cuts allows us to avoid generating a considerable number of undesirable architectures.

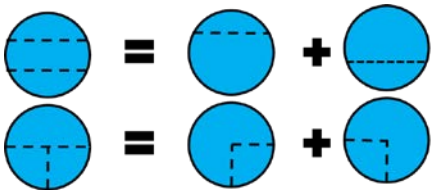

Figure 2. Boundaries between modules can be always inscribed as an overlap of a boundary between two modules.

3.2 Space Coverage

Another principle behind lowering computational cost is a population based approach. In contrast with the common multi-objective optimization (Deb et al., 2002), we do not set the objective of our algorithm to probe for non-dominated Pareto-optimal solutions. Rather, we set our objective to probe for coverage of an ever larger space of possible objective function values, as shown Figure 3. Using this population based approach, solutions that were not found useful in improving one objective might turn out to be helpful in improving the other objective. The selection criteria in our algorithm is therefore not geared toward increasing optimality of solutions but rather to enhance the diversity of values across the objective space. In so doing, however, the Pareto-frontier will be naturally obtained as the lower boundary of this diversity space. In the iterations, we keep solutions when they are sufficiently distinct from the rest of the solutions.

To implement this selection criteria, the initial space of objective function values is gridded into a matrix of subspaces and only one solution in each non-null cell of this matrix is maintained. When searching for the Pareto frontier, this is advantageous to other approaches as we keep a large diversity of solutions. This diversity leads to the outer limits of the objective function space and so Pareto

optimal solutions within a few rounds of combinations and mutation. This helps both computation time and the breadth of Pareto frontier points.

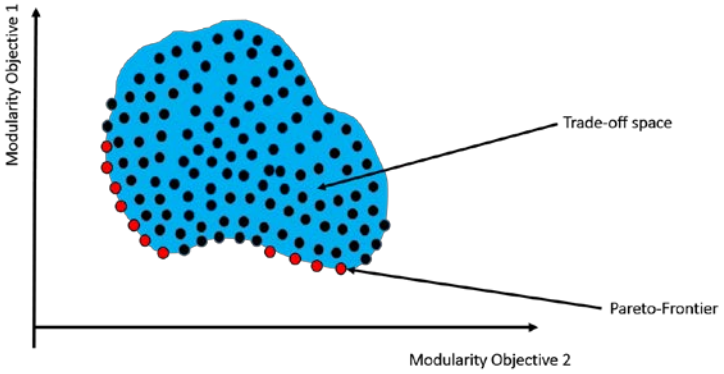

Figure 3. Visual illustration of the Pareto-frontier and trade-off space.

With a gridded space and selections made within, we then apply the second heuristic to probe the space further. We take each solution produced in exploration grid and evaluate all possible architectures that can be generated though moving one component from one module onto a neighbour. From this set, we keep only one solution to carry forward in each grid cell. In this second phase of each iteration, the grid is refined into smaller cells. The exploration and refinement process are repeated until the number of filled cells in their corresponding matrix of subspaces settles into a constant, indicating the objective function space has been adequately explored to its entire outer boundary. The outer boundary thereby includes the Pareto frontier. Having sufficiently explored the entire space we collect the minimum value of one objective for each value of other objective to depict Pareto-frontier.

3.3 DSM Pareto Optimization Algorithm

The overall structure of our algorithm is as below. The algorithm includes three parameters, *c*1, *c*2 and *n_terminate*. These can be tuned according to the type of the DSM, similar to algorithm parameters in IGTA+ or other heuristic search algorithms.

```
/* bi-sectioning stage*/
C = determine all possible cuts that can bisection the DSM;

/* exploring stage*/
architectures_set = {an architecture with no cuts};
do {
if (random_real_value1 > c1) {
New_A = take two random architectures from architectures_set and exchange their
cuts; }
 else {
A = one random architecture from architectures_set;
if (A has no cuts || (random_real_value1 > c2)) {
new_A = take a random cut from C and add it to A ;}
else { new_A = substitute a random cut from A with a random cut from C ;}}
objectives[] = objectives(New_A);
cell_coordinates[] = coordinates(Objectives);
if (grid[cell_coordinates] == empty) {
put the Objectives in grid cell;
add the architecture to architectures_set ;}
}
while (number of filled cells remains constant for n_terminate number of times)

/* refining stage*/
do {
for (A : architectures_set) {
A_set = generate all architectures possible with moving one components to one
module to other;
for (new_A : A_set) {
objectives[] = objectives(new_A);
cell_coordinates[] = coordinates(objectives);
if (Grid[cell_coordinates] == empty) {
Put the Objectives in grid cell;
Add the architecture to new_ architectures_set ;}}}
while (number of filled cells remains constant for n_terminate number of times)
```

Algorithm 1: Pseudo-code of the algorithm (without parallelization)

We also adapted our algorithm for parallel computing; to do so, the population of solutions in the grid spacing were distributed among several threads, each with separate grid matrices. Once all threads complete, results are unified into a one grid matrix. These results are the re-distributed again amongst running threads. With common multi-core processor computing systems, this parallelized approach can lead to reduction in computation time proportional to the number of cores.

A second issue is to define modularity metrics. While MFD Modularity drivers are suitable as informal objectives, they must be quantified as objective variables for clustering. We do this by associating a numerical score to each module driver. To do this, for components with similar values placed into a module we increment a score proportional to the strength of the modularity driver. For each incompatible pair of components placed within a module we similarly discount the score. The score equation can be formulated as

$$\text{Score}_k = \sum_{i=1}^{n}\sum_{j=1}^{n} S(i,j)\, C(i,j) \tag{1}$$

where S is the element wise score matrix that contains scores associated with collocation of each pair of components in the same module, and C is the elementwise cluster matrix where each of its entries denote whether or not a pair of elements are located in the same module. Here, Score matrix can be written as

$$\text{Score}_k = S(MIM(i,k), MIM(j,k)) \tag{2}$$

where $MIM(i,k)$ denotes strength of modularity driver k associated with component i. This function assigns a higher score value when the modularity driver strength of functions is higher and zero when a modernity driver is not corresponded to one of the functions. We define it simply as

$$S(x,y) = \begin{cases} x\,y & \text{if } x,y > 0 \\ 0 & \text{if } (x < 0) \text{ or } (y < 0) \end{cases} \tag{3}$$

4 CASE STUDY

To demonstrate the efficacy of the proposed algorithm, consider a practical example of a Black and Decker Dust-buster model CHV1210, a cordless handheld vacuum cleaner from (Borjesson and Hölttä-Otto, 2012). The system consists of 57 components and 89 interactions. The DSM and MIM matrices can be found in (Borjesson and Hölttä-Otto, 2012). Values for strategic parameters are shown in Table 1. Numerical values of the MIM matrix entries are defined based on MIM table values as shown in Table 2.

Table 1: Values for Algorithm Parameters

C1	0.5
C2	0.5
N_terminate	5

Table 2: MIM value quantification

Strong	1.0
Medium	0.6
Weak	0.3
None	-1.0

Grid intervals for exploration were set to 1 and 10 for the refined second phase of each iteration. For the IGTA+ algorithm, we set pow_cc = 1, and pow_bid = -2 per (Borjesson and Hölttä-Otto, 2012). The results of the search space and Pareto Frontier generated are shown in Figure 4 for both IGTA+ and our new algorithm. The new algorithm consistently generated a more accurate representation of the Pareto Frontier. Beyond improved Pareto Frontier generation, computational complexity remains

an issue with DSM search algorithms. IGTA+ took 192 hours on a computer with two Intel(r) Xeon(r) (e5-2620 v3 @ 2.40ghz) CPUs each with six physical cores (2 logical cores per physical) with 64 GB RAM to execute to generate the Pareto-frontier results shown, while our algorithm took only 24 minutes. While a multitude of different metrics can be generated as Pareto plots, the MFD drivers are shown against the 'total-cost' modularity metric in Figure 4. The 'total-cost' has been found in previous research by many to be a useful measure of complexity. As shown in Figure 4, the MFD modularity drivers are in trade-off with the total-cost metric (lower left is better). These trade-offs are more strong for the MFD metrics of styling, common unit, technical specification and planned development. Red region architecture depicts the space explored by IGTA+, blue region is explored true our algorithm, as visible our algorithm is capable of finding better non-dominated optimal architecture compared to IGTA+, in other words our algorithm is able to produce solutions are that for both objectives are higher scored compared to IGTA+.

A designer can see these particular trade-offs. An initial choice can be made, and then a designer can compare the different architectures in the neighbourhood along these particular Pareto frontiers. This will allow for a more informed decision on the trade-offs and rate of change of the strategic metric concerns with changes to the selected architecture. Such Pareto optimal architecture selection processes have been discussed elsewhere (Guo and Gershenson, 2003; Sanaei et al., 2015; Yu et al., 2007). We offer here means to efficiently compute the space of alternatives.

Essentially two main factors contribute to lower computational cost of our algorithm. First, since architectures with extra-cluster links beyond a particular threshold is undesirable, we increase the efficiency of our algorithm by heuristically limiting the generation of architectures with large number of extra-cluster links. The second factor considers the homogeneity of the population over the space of possible architectures. Our approach heuristically explores larger number of evolution paths from ordinary solutions to Pareto optimal solution.

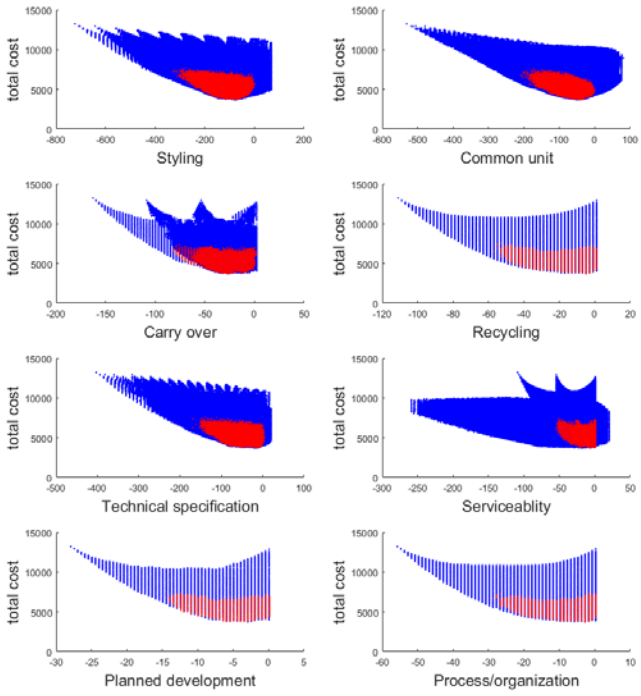

Figure 4. Various modularity driver scores versus total cost obtained using IGTA+ (red) and the new algorithm (blue). The new algorithm always produced a better Pareto Frontier.

5 CONCLUSION

In this work, we have developed design principles and implementation of a low computational-cost algorithm for generating the Pareto-optimal surface for multi-objective product architecture

modularization. This is non-trivial search, with non-convex objective functions and large space of solutions, requiring meta-heuristic search. Our approach includes two separate stages and associated heuristics for exploration.

An observation is that representing architecture as overlap of bisecting cuts is an efficient representation that helps architecture exploration by not generating high extra-cluster cost architectures. Furthermore, for multi-objective optimization, a population-oriented selection criteria that encourages diversity among individual solutions is faster at finding the outer envelope of solutions in the objective function space than spanning a weighted metric minimization.

To evaluate the algorithm performance, a set of MFD-based modularity-driver scores were defined and their trade space versus total cost were presented. The algorithm is shown to be more than two orders of magnitude faster than the previous best IGTA+ algorithm and able to produce pareto-frontier with considerable better accuracy than IGTA+, to be clear, this statement applies not to speed of optimizing a modularity metric, but rather strictly to generation of the Pareto optimal surface. Future work includes more strict computational complexity analysis and demonstrating efficiency of algorithm across multiple case studies.

6 ACKNOWLEDGEMENT

The authors would like to thank the SUTD-MIT International Design Centre (IDC, idc.sutd.edu.sg) for financial and intellectual support. Any opinions, findings, or recommendations are those of the authors and do not necessarily reflect the views of the IDC.

7 REFERENCES

Blackenfelt, M., 2001. Managing complexity by product modularisation.

Borjesson, F., Hölttä-Otto, K., 2014. A module generation algorithm for product architecture based on component interactions and strategic drivers. Res. Eng. Des. 25, 31–51.

Borjesson, F., Hölttä-Otto, K., 2012. Improved clustering algorithm for design structure matrix, in: ASME 2012 IDETC/CIE, pp. 921–930.

Dahmus, J.B., Gonzalez-Zugasti, J.P., Otto, K.N., 2001. Modular product architecture. Des. Stud. 22, 409–424.

Deb, K., Pratap, A., Agarwal, S., Meyarivan, T., 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans. Evol. Comput. 6, 182–197.

Du, X., Jiao, J., Tseng, M.M., 2001. Architecture of product family: fundamentals and methodology. Concurr. Eng. 9, 309–325.

Eppinger, S.D., Browning, T.R., 2012. Design structure matrix methods and applications. MIT press.

Ericsson, A., Erixon, G., 1999. Controlling design variants: modular product platforms. Society of Manufacturing Engineers.

Gershenson, J.K., Prasad, G.J., Zhang, Y., 2003. Product modularity: definitions and benefits. J. Eng. Des. 14, 295–313.

Gu, P., Sosale, S., 1999. Product modularization for life cycle engineering. Robot. Comput.-Integr. Manuf. 15, 387–401.

Guo, F., Gershenson, J.K., 2003. Comparison of modular measurement methods based on consistency analysis and sensitivity analysis, in: ASME 2003 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference. American Society of Mechanical Engineers, pp. 393–401.

Gutierrez, C.I., 1998. Integration analysis of product architecture to support effective team co-location. Massachusetts Institute of Technology.

Hölttä, K.M., Otto, K.N., 2005. Incorporating design effort complexity measures in product architectural design and assessment. Des. Stud. 26, 463–485.

Holtta, K.M., Otto, K.N., 2003. Incorporating design complexity measures in architectural assessment, in: ASME 2003 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference. American Society of Mechanical Engineers, pp. 525–532.

Holtta, K.M., Salonen, M.P., 2003. Comparing three different modularity methods, in: ASME 2003 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference. American Society of Mechanical Engineers, pp. 533–541.
Hölttä-Otto, K., Chiriac, N.A., Lysy, D., Suk Suh, E., 2012. Comparative analysis of coupling modularity metrics. J. Eng. Des. 23, 790–806.
Hölttä-Otto, K., De Weck, O., 2007. Degree of modularity in engineering systems and products with technical and business constraints. Concurr. Eng. 15, 113–126.
Kimura, F., Kato, S., Hata, T., Masuda, T., 2001. Product modularization for parts reuse in inverse manufacturing. CIRP Ann.-Manuf. Technol. 50, 89–92.
McAdams, D.A., Stone, R.B., Wood, K.L., 1999. Functional interdependence and product similarity based on customer needs. Res. Eng. Des. 11, 1–19.
Osman, I.H., Kelly, J.P., 1996. Meta-heuristics: an overview, in: Meta-Heuristics. Springer, pp. 1–21.
Otto, K., Wood, K., 2001. Product Design: Techniques in reverse engineering, systematic design, and new product development. Prentice Hall, New York.
Rechtin, E., 1991. Systems architecting: Creating and building complex systems. Prentice Hall Englewood Cliffs, NJ.
Rogers, G.G., Bottaci, L., 1997. Modular production systems: a new manufacturing paradigm. J. Intell. Manuf. 8, 147–156.
Salonen, M., Holtta-Otto, K., Otto, K., 2008. Effecting product reliability and life cycle costs with early design phase product architecture decisions. Int. J. Prod. Dev. 5, 109–124.
Sanaei, R., Otto, K., Hölttä-Otto, K., Luo, J., 2015. Trade-Off Analysis of System Architecture Modularity Using Design Structure Matrix, in: ASME 2015 IDETC/CIE., p. V02BT03A037-V02BT03A037.
Sanai, R., Otto, K., Wood, K., Hölltä-Otto, K., 2016. Trade-offs Among System Architecture Modularity Criteria.
Sanchez, R., Mahoney, J.T., 1996. Modularity, flexibility, and knowledge management in product and organization design. Strateg. Manag. J. 17, 63–76.
Sharman, D.M., Yassine, A.A., 2007. Architectural valuation using the design structure matrix and real options theory. Concurr. Eng. 15, 157–173.
Sosa, M.E., Eppinger, S.D., Rowles, C.M., 2003. Identifying modular and integrative systems and their impact on design team interactions. J. Mech. Des. 125, 240–252.
Stone, R.B., Wood, K.L., 2000. Development of a functional basis for design. J. Mech. Des. 122, 359–370.
Stone, R.B., Wood, K.L., Crawford, R.H., 2000. Using quantitative functional models to develop product architectures. Des. Stud. 21, 239–260.
Stone, R.B., Wood, K.L., Crawford, R.H., 1998. A heuristic method to identify modules from a functional description of a product, in: Proceedings of DETC98. pp. 1–11.
Thebeau, R.E., 2001. Knowledge management of system interfaces and interactions from product development processes. Massachusetts Institute of Technology.
Ulrich, K., 1994. Fundamentals of product modularity, in: Management of Design. Springer, pp. 219–231.
Van Beek, T.J., Erden, M.S., Tomiyama, T., 2010. Modular design of mechatronic systems with function modeling. Mechatronics 20, 850–863.
Van Wie, M.J., Greer, J.L., Campbell, M.I., Stone, R.B., Wood, K.L., 2001. Interfaces and product architecture, in: ASME DETC01/DTM.
Wolpert, D.H., Macready, W.G., 1997. No free lunch theorems for optimization. IEEE Trans. Evol. Comput. 1, 67–82.
Yu, T.-L., Yassine, A.A., Goldberg, D.E., 2007. An information theoretic method for developing modular architectures using genetic algorithms. Res. Eng. Des. 18, 91–109.
Yu, T.-L., Yassine, A.A., Goldberg, D.E., 2003. A genetic algorithm for developing modular product architectures, in: ASME 2003 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference. American Society of Mechanical Engineers, pp. 515–524.
Zamirowski, E.J., Otto, K.N., 1999. Identifying product portfolio architecture modularity using function and variety heuristics, in: Proceedings of the 11th International Conference on Design

Theory and Methodology ASME Design Engineering Technical Conferences, Las Vegas, NV, Paper No. DETC99/DTM-8790.