Pattern Identification in Pareto-Set Approximations

Tamara Ulrich, Dimo Brockhoff and Eckart Zitzler
Computer Engineering and Networks Lab
ETH Zurich
8092 Zurich, Switzerland
firstname.lastname@tik.ee.ethz.ch

ABSTRACT

In a multiobjective setting, evolutionary algorithms can be used to generate a set of compromise solutions. This makes decision making easier for the user as he has alternative solutions at hand which he can directly compare. However, if the number of solutions and the number of decision variables which define the solutions are large, such an analysis may be difficult and corresponding tools are desirable to support a human in separating relevant from irrelevant information.

In this paper, we present a method to extract structural information from Pareto-set approximations which offers the possibility to present and visualize the trade-off surface in a compressed form. The main idea is to identify modules of decision variables that are strongly related to each other. Thereby, the set of decision variables can be reduced to a smaller number of significant modules. Furthermore, at the same time the solutions are grouped in a hierarchical manner according to their module similarity. Overall, the output is a dendrogram where the leaves are the solutions and the nodes are annotated with modules. As will be shown on knapsack problem instances and a network processor design application, this method can be highly useful to reveal hidden structures in compromise solution sets.

Categories and Subject Descriptors

I.5.2 [**Pattern Recognition**]: Design Methodology—*Pattern analysis*; I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*Heuristic methods*

General Terms

Algorithms

Keywords

decision making, heuristics, multi-objective optimization, representations

GECCO'08 July 12-16, 2008, Atlanta, Georgia, USA.

1. INTRODUCTION

In a multiobjective optimization scenario, it is often useful to approximate the set of Pareto-optimal solutions in order to learn about the underlying problem and to achieve information that provides a better basis for decision making. By presenting such a set of compromise solutions, the so-called Pareto-set approximation, to the user, he can not only study the relationships among the objectives, but also gain insights about the inherent structure of the problem. This requires, though, that the number of solutions, the number of problem parameters, and the number of objectives are reasonably small. Nowadays, increased computing resources allow to cope with problems that have more and more decision variables and objectives, e.g., see [15, 9]. Therefore, in practice all three entities can become large and tools are needed that help the decision maker in analyzing the trade-off surface.

The issue of dealing with many objective functions has been recently addressed in a few studies (see [2] for an overview). Different methods have been proposed to reduce the number of objective functions by omitting certain criteria such that the resulting error is minimized; this can be helpful both for assisting in decision making and for speeding up the search. The problem of many decision variables has been mainly studied in the context of search, e.g., [17, 7]; only few contributions exist in the context of Pareto-set analysis, e.g., [12]. However, Deb and Srinivasan [3] have shown that important structural information in the decision space may be contained within a non-dominated set.

In this paper, the focus lies on decision making and on how an automated technique can be used to achieve a more compact representation of a Pareto-set approximation. Here, we aim at reducing along two dimensions simultaneously: the number of decision variables and the number of solutions. The first aspect is tackled using biclustering techniques in order to find interesting modules of decision variables which have similar values and are jointly represented in many solutions. As to the second aspect, a greedy strategy is proposed to group solutions containing similar modules hierarchically. In detail, the paper contains the following main contributions:

- we formalize the general problem of module identification and solution grouping (as a first study, we only consider Boolean decision variables here);
- we propose two contrary methods to generate dendrograms of a solution set that are annotated by modules;
- we validate and empirically investigate the proposed methods for instances of the 0-1 knapsack problem;

- we demonstrate the usefulness of the proposed approach for a network processor design problem.

The remainder of the paper is structured as follows. After discussing related work in Sec. 2, we formalize the two-step decision space reduction in Sec. 3. The proposed algorithms are presented in Sec. 4, and Sec. 5 shows the results of the algorithm comparison. Sec. 6 applies the proposed decision space reduction approach to a network processor design problem and Sec. 7 concludes the paper.

2. RELATED WORK

The approach of pattern identification in Pareto-set approximations presented in this paper is strongly related to the concepts of building blocks as well as to biclustering. In this work, we are aiming at finding modules or building blocks [8] of the decision variables; in other words we would like to identify structure in Pareto-optimal sets or approximations thereof.

Building blocks have already been used explicitly *during* both single-objective search, e.g., in the messy GA [5], and multiobjective search [17]. In the messy GA, promising building blocks are generated prior to the search. Here, we argue that an automated identification of those building blocks in a given Pareto-set approximation makes also sense *after* the search to assist in decision making. Unlike in the messy GA, we would like to generate building blocks based on problem specific information that is provided in the decision space. The consideration of the decision space is in fact crucial in the case of multiobjective optimization, as was indicated recently by Preuss et al. [14].

The methods proposed in this thesis try to identify sets of decision variables that exhibit homogeneous behavior over a large number of solutions. This approach corresponds to the concept of biclustering. Biclustering is a recent extension of standard clustering which, roughly speaking, aims at finding large homogeneous submatrices in a matrix, the so-called biclusters. Biclustering has become popular especially in computational biology, see [11] for a survey. Biclustering methods mainly differ in the definition of homogeneity, the distribution of the biclusters found and the strategies which are used to find the biclusters. One of the first biclustering algorithms presented in the literature was the one of Hartigan [6] which is, due to its simplicity, also used in this paper. However, Hartigan's algorithm is not able to find biclusters that overlap which is its main drawback. An algorithm which not only allows the biclusters to overlap but also finds the exhaustive set of all biclusters[1] is Bimax [13]. Such an exhaustive search, however, is only applicable for small and/or sparse matrices.

3. PROBLEM FORMULATION

Suppose we have a multiobjective maximization problem $f : X \to Z$ with $Z \subseteq \mathbb{R}^k$ and a multiobjective evolutionary algorithm that generates a Pareto-set approximation. Such a Pareto-set approximation can be considered as a set of *decision vectors* $\{x^1, \ldots, x^m\} \subseteq X$ that are mutually non-dominated. A decision maker can analyze these decision vectors and needs to choose one out of them. In this paper, we consider only binary decision vectors, i.e., $X = \{0,1\}^n$ and represent a Pareto-set approximation as a *decision matrix* $\Xi \in \mathcal{M}_{m,n}(\{0,1\})$ where $\mathcal{M}_{m,n}(\{0,1\})$ is the set of binary matrices with m rows and n columns.

[1]Except for biclusters that are entirely contained in larger ones.

DEFINITION 1. *A* decision matrix $\Xi = (\xi_{i,j})_{m \times n}$ *is a matrix with* n *columns and* m *rows that is composed of the decision vectors* $x^r = (\xi_{r,1}, \ldots, \xi_{r,n})$ *of* m *solutions (*$1 \leq r \leq m$*).*

In practice, two main problems emerge. The first problem is that there are too many decision variables. The methods proposed in this thesis tackle this problem by merging the decision variables into so-called *modules*.

DEFINITION 2 (MODULE). *A* module *is a subset* $S \subseteq \{1, \ldots, n\}$ *of the decision variables.*

These modules are then used to generate a new reduced representation of the decision variables.

The second problem is that there are too many solutions. This problem is tackled by grouping solutions hierarchically. Sec. 4 introduces a method to generate such a grouping based on modules. Both the problem of finding the best reduced representation and the problem of grouping the solutions are formalized in the following.

3.1 Transformation to a New Representation

When identifying modules, the goal is to find a small set of large modules. By representing modules instead of single decision variables, a reduced representation of a decision matrix can be achieved. More precisely, given a set of modules $S = \{S_1, \ldots, S_l\}$, we would like to transform the decision matrix $\Xi \in \mathcal{M}_{m,n}(\{0,1\})$ into a new representation, the *module matrix* Υ wherein the rows correspond to the original solutions in Ξ and the columns correspond to the modules in S. For a certain solution x^r, the ith bit in the new representation y^r is set to 1 if and only if the original representation contains the module S_i, i.e., if and only if all decision variables belonging to S_i are set to 1 in x^r.

DEFINITION 3. *Given a decision matrix* $\Xi = (\xi_{i,j})_{m \times n} \in \mathcal{M}_{m,n}(\{0,1\})$ *and a set of modules* $S = \{S_1, \ldots, S_l\}$*, the function* $T_{\Xi \to \Upsilon}(\Xi, S)$ *yields a corresponding module matrix* $\Upsilon = (\upsilon_{i,j})_{m \times l}$*, which is defined as* $\upsilon_{r,c} = 1 \Leftrightarrow \forall i \in S_c : \xi_{r,i} = 1$ *for all* $1 \leq r \leq m$ *and* $1 \leq c \leq n$*. Each row of* Υ *is called a* module vector*.*

Note that we here assume that whenever a module is selected, all contained decision variables are set to 1. In general, one could consider an arbitrary variable assignment representing the module; for reasons of simplicity, we do not consider this further in this paper.

EXAMPLE 1. *Consider a decision matrix* Ξ *with five solutions and decision vectors of length* 5 *as depicted on the left of Fig. 1. In addition, the module set* S *consists of three modules* $S_1 = \{1,2,3\}$*,* $S_2 = \{2,3,4\}$*, and* $S_3 = \{4,5\}$*. The above defined transformation* $T_{\Xi \to \Upsilon}$ *maps the decision matrix* Ξ *to the new representation* $\Upsilon = T_{\Xi \to \Upsilon}(\Xi, S)$ *as shown on the right of Fig. 1. For example, the decision vector* x^3 *has ones at the positions 1 to 4 and therefore contains both modules* $S_1 = \{1,2,3\}$ *and* $S_2 = \{2,3,4\}$ *but not module* S_3 *since the fifth bit is not set to 1. Therefore, its corresponding module vector* y^3 *in* Υ *contains ones at the positions* 1 *and* 2 *and a zero at position* 3*.*

Note that in the above example, the module matrix can cover all 1s in the original decision matrix. In general, this is not the case as the following example shows.

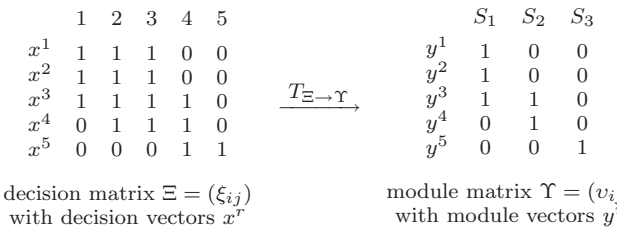

Figure 1: Illustration of the decision matrix/module matrix concept for the given modules $S_1 = \{1, 2, 3\}$, $S_2 = \{2, 3, 4\}$, and $S_3 = \{4, 5\}$.

EXAMPLE 2. *Consider the decision vector x^1 in Fig. 2 and the same modules as in Example 1. Since x^1 only contains module S_3 but not S_1 and S_2, the bit $\xi_{1,2}$ cannot be reconstructed with the module representation Υ. Therefore with the transformation $T_{\Xi\to\Upsilon}$ information is lost.*

To measure the information loss described in the previous example, we interpret a module matrix again as a decision matrix by retransforming it with the following function.

DEFINITION 4. *Given a set $S = \{S_1, \ldots, S_l\}$ of modules, a module vector $y^r = (\upsilon_{r,1}, \ldots, \upsilon_{r,l})$ can be interpreted as the decision vector $T_{\Upsilon\to\Xi}(y^r, S) = (\xi_{r,1}, \ldots, \xi_{r,n})$ where a bit $\xi_{r,c}$ is set to 1 if at least one entry $\upsilon_{r,i}$ in y^r is set to 1 for which the module S_i contains the column c, i.e., $\xi_{r,c} = 1 \Leftrightarrow \exists S_i \in S : \upsilon_{r,i} = 1 \wedge c \in S_i$ for all $1 \le r \le m$ and $1 \le c \le n$.*

When reducing the decision matrix to the module matrix, we want to achieve the smallest representation while most of the information has to be preserved. More formally, we assess a certain transformation by computing an error function $e(\Xi, T_{\Upsilon\to\Xi}(T_{\Xi\to\Upsilon}(\Xi, S), S))$ between the original decision matrix Ξ and the corresponding retransformed module matrix $T_{\Upsilon\to\Xi}(T_{\Xi\to\Upsilon}(\Xi, S), S)$. This error can be defined with respect to both decision space and objective space. Here, we use the following two error functions:

DEFINITION 5. *Let $\Xi = (\xi_{i,j})_{m\times n} \in \mathcal{M}_{m,n}(\{0,1\})$ and $\Xi^T = (\xi^T_{i,j})_{m\times n} \in \mathcal{M}_{m,n}(\{0,1\})$ be two decision matrices. Then, one possible error function with respect to decision space is the Hamming distance between the matrices:*

$$e_{\text{dec}}(\Xi, \Xi^T) := \sum_{1\le i\le m} \sum_{1\le j\le n} |\xi_{i,j} - \xi^T_{i,j}|.$$

An error function with respect to to objective space can be defined as

$$e_{\text{obj}}(\Xi, \Xi^T) := \sum_{1\le i\le m} I_\varepsilon\left(f((\xi^T_{i,1}, \ldots, \xi^T_{i,n})), f((\xi_{i,1}, \ldots, \xi_{i,n}))\right)$$

where I_ε is the binary additive epsilon indicator of [21]. Note that other quality indicators like the hypervolume indicator in [21] can be used as well. The second error function gives an idea of the change in objective vector values if the new module representation is used.

Now, we can state the problem of finding a best set of modules according to a given error function:

PROBLEM 1 (BI-OBJECTIVE MODULE SELECTION). *Let $\Xi \in \mathcal{M}_{m,n}(\{0,1\})$ be a decision matrix and $e : \mathcal{M}_{m,n}(\{0,1\}) \times \mathcal{M}_{m,n}(\{0,1\}) \to \mathbb{R}$ an error function that computes an error between two arbitrary decision matrices. Then, the bi-objective problem of* simultaneously selecting a module set and minimizing the number of modules *can be stated as finding a set $S = \{S_1, \ldots, S_l\}$ such that both the number of modules l and the error $e(\Xi, T_{\Upsilon\to\Xi}(T_{\Xi\to\Upsilon}(\Xi, S), S))$ are minimized.*

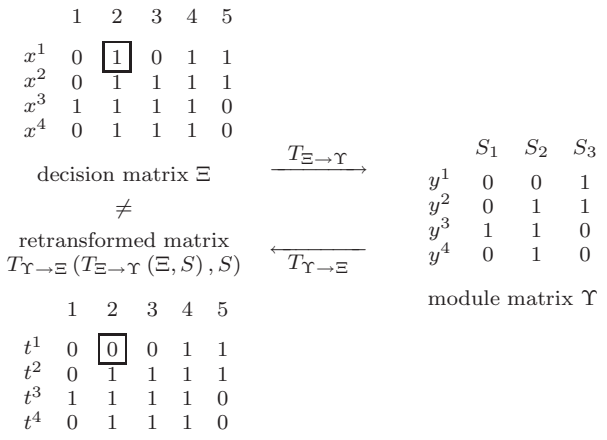

Figure 2: An example where the retransformation does not yield the original decision matrix. The modules are defined as $S_1 = \{1, 2, 3\}$, $S_2 = \{2, 3, 4\}$, and $S_3 = \{4, 5\}$.

This problem is $\mathcal{NP}$-hard; the proof is given in the appendix. Methods to tackle this module selection problem will be presented in Sec. 4.

3.2 Grouping Solutions by Using Structure Information

Given a set of modules, we would like to reduce the number of solutions by merging them into hierarchical groups. The goal is to generate groups whose solutions are as similar as possible. This in general corresponds to the task of clustering. Instead of setting the number of groups a priori, we would like to be able to traverse the group hierarchy from the largest group, which contains all solutions, to the smallest groups where each group consists of only one solution. To achieve this, we propose to use dendrograms to represent the grouping structure. The resulting groups should strongly depend on the modules found, such that each group can be uniquely defined by a sequence of modules that are selected in this group. To this end, module-annotated dendrograms are introduced.

In general, a dendrogram is a binary tree which can be used to represent a hierarchically organized grouping structure. An example is given in Fig. 3. The nodes are distributed on so-called levels, i.e., each node has a fixed distance from the root. In a module-annotated dendrogram, each level has exactly one node, reflecting the order in which modules are selected for the grouping. Each node is associated with one module, where solutions containing the module all belong to the left branch of the node, and solutions that do not contain the module belong to the right branch. The leaves represent the rows of the decision matrix, i.e., the solutions in a Pareto-set approximation. The branches represent groups which contain all solutions (leaves) below

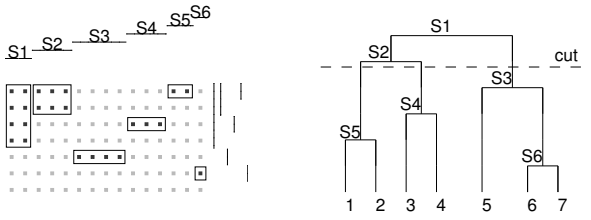

Figure 3: Example of a dendrogram with additional module annotations (right) for a given decision matrix (left). The solutions are denoted by the numbers from 1 to 7 and the modules by S_1 to S_6. The vertical lines on the right of the decision matrix indicate the corresponding groups.

that branch. In general, solutions and groups of solutions which lie close to each other have many modules in common and therefore have a high similarity.

We consider the goal of identifying the dendrogram that minimizes the distances of the solutions within the groups. As a distance measure of a group $G \subseteq \{1,\dots,m\}$ of solutions, we use the average pairwise Hamming distance $s(G) := 1/\binom{|G|}{2} \sum_{r,s\in G} d_{\mathrm{H}}(x^r, x^s)$ where the Hamming distance between two points $x^r = (x^r_1,\dots,x^r_n)$ and $x^s = (x^s_1,\dots,x^s_n)$ is defined as $d_{\mathrm{H}}(x^r, x^s) = \sum_{1\le j\le n} |x^r_j - x^s_j|$. For evaluating an entire dendrogram, we use the intra-group distance measure as defined above averaged over all groups in a level cut and averaged over all these cuts. A level cut divides the dendrogram horizontally, such that with each level cut a set of groups is associated. For example, the level cut between S_2 and S_3 in Fig. 3 contains three groups: The one where all solutions contain S_1 and S_2 (left subtree), one where all solutions contain S_1 but not S_2 (middle) and the third where all solutions neither contain S_1 nor S_2 (right subtree).

Definition 6. *As* distance measure s *of a dendrogram D with the level cuts $C_1,\dots,C_m \subseteq 2^{\{1,\dots,m\}}$, where each level cut C_i is a set of groups $C_i = \{G_{i,1},\dots,G_{i,|C_i|}\}$ ($G_{i,j} \in \{1,\dots,m\}$) we propose the average pairwise intra-group Hamming distance, averaged over all groups and all cuts. The number of groups associated with a cut is equal to the number of intersections between the cut and the dendrogram branches.*

$$s(D) := \frac{1}{m} \sum_{1\le i\le m} \frac{1}{|C_i|} \sum_{1\le j\le |C_i|} \frac{1}{\binom{|G_j|}{2}} \sum_{r,s\in G_{i,j}} d_{\mathrm{H}}(x^r, x^s).$$

Overall, this leads to the following problem which has been shown to be $\mathcal{NP}$-hard [10].

Problem 2 (Finding the Optimal Dendrogram). *Given a decision matrix Ξ, the problem of* finding the optimal module-annotated dendrogram *corresponds to finding the dendrogram D with the lowest distance measure $s(D)$ as defined in Definition 6.*

4. APPROACHES

Since the two problems presented in the previous section are $\mathcal{NP}$-hard, we propose corresponding heuristics in the following. More precisely, we propose (i) two approaches based on biclustering for approximating the module selection problem and (ii) a method to construct a dendrogram on that basis to deal with Problem 2.

4.1 Module finding

As described in Sec. 3 we would like to find modules that exhibit homogeneous behavior over many solutions. This problem corresponds to the task of biclustering. In the following, a bicluster is defined as a submatrix of Ξ that only contains ones. Each of these biclusters forms a module consisting of the bicluster's columns. In Sec. 2 we have decided to use two exemplary biclustering algorithms: Hartigan's algorithm [6] and Bimax [13].

Both algorithms have their advantages and drawbacks but due to their complementary behavior, we selected them as representative examples of biclustering algorithms. The Hartigan algorithm is the first proposed biclustering algorithm, and many other algorithms are based on its principles, cf. [11]; it is simple and fast. In contrast to the Bimax algorithm, it limits the number of possible biclusters substantially as it does not find overlapping biclusters. The Bimax algorithm, however, finds all possible inclusion maximal biclusters, i.e., all biclusters which are not contained in larger ones. As the number of all biclusters is in general exponential in the matrix size this algorithm is impractical for larger matrices.

Hartigan's Algorithm: Hartigan's algorithm is based on a simple divide-and-conquer strategy; it iteratively divides the decision matrix into smaller submatrices. Due to this strategy, the order of the rows and columns of the decision matrix is fixed as soon as the splitting starts. The matrix therefore has to be sorted prior to algorithm execution. To be able to identify large biclusters, an appropriate sorting measure is essential.

In this thesis, we use two criteria for the initial sorting: one sorts according to the Hamming distances in decision space and the other sorts according to the objective space values. The first criterion places the two solutions with the highest Hamming distance as first and last row, making them the upper and lower border solution. It then iteratively selects the solution with the smallest Hamming distance to either border solution, places it next to this border solution and makes it the new respective border solution. The second criterion is restricted to two-objective problems; it simply sorts the solutions in the decision matrix according to their values of the first objective.

After sorting, the iterative splitting of the matrix takes place. In each step, the theoretical best split for each existing submatrix is calculated and the best overall split is performed by splitting one of the existing submatrices into two new submatrices. The algorithm stops as soon as each submatrix contains only ones or only zeros. As a splitting measure for Hartigan's algorithm, we take the following *percentage split measure*, defined as

$$Q(M_1, M_2) = \left| \frac{\#\text{ ones in } M_1}{|M_1|} - \frac{\#\text{ ones in } M_2}{|M_2|} \right|$$

where M_1 and M_2 are the two submatrices resulting from the split. This split measure has to be maximized in order to find the best split.

Bimax: The recursive Bimax algorithm performs an exhaustive search for the set of all biclusters using a branch-and-bound strategy. Even for reasonably sized matrices, the number of biclusters found can become very high. Therefore, we use a heuristic method to prune the set of biclusters found. The pruning method iteratively selects the bicluster which covers most of the remaining 1s. The remaining 1s are

defined as the 1s not yet covered by any selected bicluster. This iteration stops if either a predefined number of selected biclusters is reached or all 1s of the matrix are covered by the selected biclusters.

4.2 Grouping Solutions Within Dendrogram

To create a module-annotated dendrogram, hierarchical clustering could be used on the reduced representation, in which case each module would contribute equally to the grouping. Here, however, we would like a group to be defined by the sequence of modules that are selected in all solutions of the group. We propose the following simple approach. The grouping starts according to the largest bicluster. This bicluster divides the solutions into two groups, namely those solutions which contain the module given by the bicluster and those that do not. This already defines the root of the dendrogram completely. Then, the next largest bicluster has to be selected where the size of a bicluster is defined as the number of ones covered by this bicluster which are not covered by any previously chosen bicluster. The generation of the dendrogram stops if all groups contain only one solution.

5. EXPERIMENTAL VALIDATION

In this section, we address two questions: (i) are the algorithms successful in finding meaningful groups, and (ii) are there interesting structures present in Pareto-optimal sets and approximations thereof. These aspects are studied on the basis of the bi-objective 0-1-knapsack problem [20].

5.1 Proof-of-principle Results on Well-structured Matrices

To show that both methods presented in the last section can find known structures in a given decision matrix Ξ, we implant random biclusters, each defining a particular module, into a matrix[2] and analyze the capability of the two biclustering algorithms to find the corresponding modules. In detail, biclusters that contain the same solutions are merged to constitute one bicluster beforehand. Each of these enlarged implanted biclusters corresponds to a module which contains all columns the bicluster contains. To check whether both Hartigan's algorithm and Bimax find these modules, we use the following measure. For each implanted module, we compute the module found by the biclustering algorithms that matches the implanted module best, i.e., which has the highest ratio of shared columns to the union of both column sets. The average of these best ratios over all implanted modules indicates the percentage of implanted modules that are covered by the automatically identified modules.

The results for different matrix sizes and different densities are shown in Table 1. Two major observations can be made. First, Bimax finds more of the implanted structure than Hartigan due to its exhaustive search for biclusters. The covering is not 100% for Bimax because it only finds inclusion maximal biclusters, which can be larger than the implanted biclusters. Second, the results for the sparse matrices are in all cases better than for the dense matrices. This can be explained by the high number of implanted biclusters in the dense matrices and the issue that even Bimax does not find all of these biclusters since the pruning heuristic of Sec. 4 was used.

matrix size	percentage of ones in matrices	percentage of covering	
		Hartigan	Bimax
50x50	20	69.72	99.11
50x50	50	61.11	82.74
100x100	20	75.22	93.73
100x100	50	49.97	71.72
300x300	20	51.65	75.92
300x300	50	35.56	n/a

Table 1: Percentage of modules found in structured random matrices that are covered by implanted modules. Note that the number of biclusters found by Bimax on the dense 300x300 matrix was already too large, i.e., its running time longer than one day.

5.2 Pareto-Optimal Sets Contain Structure

We would now like to show that Pareto-optimal sets actually contain structure. As a test case, the knapsack problem is chosen, as its Pareto-optimal set can be calculated exactly using an integer linear programming solver. If the hypothesis that a Pareto-optimal set actually contains structure holds, then the corresponding decision matrix should contain larger and fewer modules then a random matrix; this, in turn, should be reflected in a smaller error as defined in Definition 5.

Here, we compare the Pareto-optimal sets of 11 different bi-objective knapsack instances including 100 items with 11 randomly generated matrices of similar size[3] with respect to the structure that is found by the two proposed methods based on Hartigan's algorithm and Bimax. The random matrices are generated by setting every entry to '1' independently with probability 0.5; the solution's objective vectors are also randomly chosen by assigning randomly generated profits and weights to the 0-1-knapsack problem.

The results as depicted in Fig. 4 indicate that the Pareto-optimal fronts contain more structure than the random matrices. In detail, both Bimax and Hartigan find modules that yield smaller errors for the Pareto-optimal fronts than for the random matrices if the same number of modules are taken into account. Note that although the objective space values are not taken into account by either method, the error in objective space is significantly smaller for Pareto-optimal fronts than for random matrices. Furthermore, we have to note that Bimax was not applicable on the random matrices since the number of biclusters found is too high. However, Bimax finds better modules in the structured Pareto-optimal sets yielding a lower error than those found by Hartigan's algorithm. An error of zero is already reached with about 50 modules which results in a reduction of the decision variables of about 50% in the corresponding module matrix.

[2]To this end, random biclusters are generated until the desired number of ones is reached. The biclusters are then placed in the matrix randomly in order of their sizes—starting with the largest—with the restriction that biclusters cannot overlap.

[3]The size is chosen by calculating the average length and width of the knapsack Pareto-optimal sets. In this case, there are on average 150 solutions and 55 items which are neither contained in all nor in none of the solutions. Note that we are not interested in decision variables that are contained in all or no solution. Therefore, such columns are deleted prior to module finding.

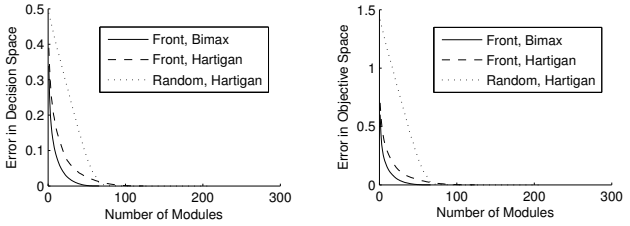

Figure 4: Comparison between Pareto-optimal sets and random matrices with respect to error function e_{dec} (left) and e_{obj} (right) averaged over 11 instances. The error is plotted against the number of modules taken into account if the modules are chosen as in the dendrogram, i.e., according to their size—starting with the largest.

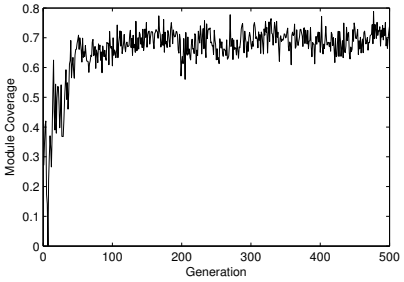

Figure 5: Average coverage of the modules found by Hartigan's algorithm on SPEA2's current non-dominated individuals plotted over time.

5.3 Progress of Structure During Search

To study the change of the structure of the Pareto-optimal set approximation *during* the run of a multiobjective evolutionary algorithm, we apply the SPEA2 algorithm [19] to one of the 0-1-knapsack instances of the previous section[4]. This is the first step towards an automated detection of problem structure to speed up the search. However, it remains future work to study an online search space reduction in depth. Figure 5 shows the progress of similarity between the population's modules and the modules contained in the Pareto-optimal set over time. In this case, Hartigan's algorithm was applied both to the sets of non-dominated solutions in each generation and on the Pareto-optimal set itself to find the contained modules. The similarity of modules is defined as the deviation between the two column sets as described in Sec. 5.1.

Figure 5 shows the trend of the covering over time. As expected, the modules found in the population become more and more similar to the ones contained in the Pareto-optimal set as the population converges to the Pareto-optimal set, although the fluctuations of the similarity are quite large.

5.4 Running Times

The Bimax algorithm has a worst-case running time which is exponential in the matrix size. This is mainly due to the number of biclusters found, which limits the usage of Bimax.

[4]For SPEA2, the implementation from the PISA toolbox with standard parameter values is used [1]. The population size is set to 300 and the knapsack instance has 100 items.

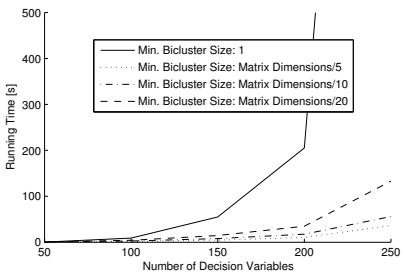

Figure 6: Running time of Hartigan's algorithm on Pareto-optimal sets of the 0-1-knapsack problem for different input sizes and different minimal bicluster sizes. Note that the number of solutions in the Pareto-optimal sets is 142 on average for the 100 item instances and 344 for the 250 item instances.

For example, the decision matrix of size 701×123 containing the solutions of a Pareto-optimal set from a 250 items knapsack instance produce more than 2 GB of data. One way to reduce this huge amount of data is to restrict the minimum bicluster size. However, this cannot solve the problem completely. Although structured matrices of size 300×300 can be processed, Bimax needs more than one day on an AMD 64bit linux machine with 4 cores and 2.6GHz to process random matrices of the same size. The usage of Bimax is therefore limited to small instances. However, it served as a reference method that yields—due to its exhaustive search for biclusters—better results than Hartigan's algorithm.

For Hartigan's algorithm, a similar restriction on the minimum bicluster size can be used which makes the algorithm applicable to matrices of reasonable size, see Fig. 6. For example, the computation of the modules within a Pareto-optimal set of a 250 item knapsack instance with 344 solutions takes about one minute on the AMD linux machine mentioned above if the minimum bicluster size is set to 10% of the matrix dimensions.

6. APPLICATION

In this section, we apply the proposed approaches of module identification and grouping of solutions to two examples to show what can be gained from an analysis of the structure in Pareto-optimal sets.

6.1 Knapsack Problem

For the two-objective 0-1-knapsack problem, we focus on the grouping according to both similarity in decision space and objective space. This is not directly provided by the proposed approaches but can be gained indirectly by sorting the decision matrix within the Hartigan framework according to objective space and doing the grouping according to decision space. For sets of non-dominated solutions of a two-objective problem, the sorting of the decision matrix according to objective space values can be achieved without loss of generality by sorting according to the values of the first objective. Figure 7 shows a grouping example for a Pareto-set approximation of a 0-1-knapsack problem instance with 100 items, generated by SPEA2 as in the previous section, using the settings of [20].

For illustrating the similarities within the groups, we can additionally plot the profit-to-weight ratios of the items of

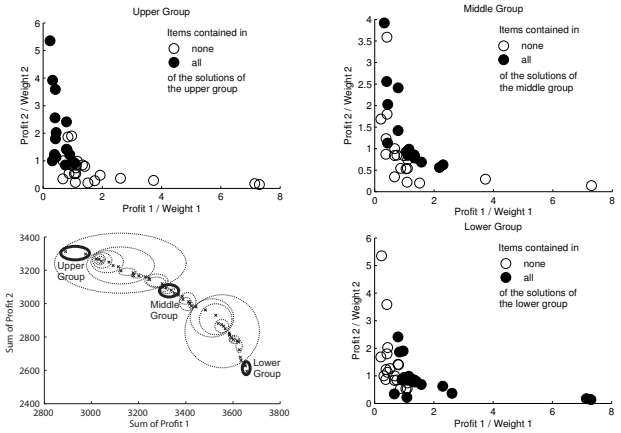

Figure 7: Grouping of a Pareto-optimal set approximation for the knapsack problem with 300 solutions and 11 groups (lower left) and item representation of three exemplary groups: the group with highest f_1 values is shown in the lower right figure, the group with the highest f_2 values is shown in the upper left figure, and the upper right figure shows a group with intermediate objective values. The grouping is done with modules of Hartigan's algorithm.

the knapsack instance and indicate for each group which items are included in all solutions of the considered group (black), and included in no solution of the group (white). Figure 7 shows the profit-to-weight ratio plots for three exemplary groups of the investigated knapsack instance. For clarity, items that are contained in all or in no solutions of the entire Pareto-optimal set are not plotted. Interestingly, the analyzed Pareto-optimal set contains structure within both decision and objective space: solutions which are neighbored in objective space also show similarities in their decision vectors. Solutions located on the same extreme of the Pareto-optimal frontier have similar items selected whereas solutions on opposite extremes have complementary decision vectors; for solutions that have a high f_1 value, items with high f_1 profit are selected, whereas solutions with high f_2 values contain more items with high f_2 profit.

6.2 Network Processor Design

As a second application, we choose the problem of a network processor design as described in [16] and as provided in the PISA framework [1]. The problem is to optimize the architecture of packet processing devices with respect to the two objectives performance and cost. In more detail, components of the processor have to be chosen and computing tasks have to be assigned to these components afterwards. To investigate the underlying structure of this problem, we use the multiobjective optimizer IBEA [18] to generate a Pareto-optimal set approximation. To this end, the algorithm is run with a population size of 150 for 300 generations. Only the 33 non-dominated solutions found are used in the analysis based on Hartigan's algorithm.

Figure 8 illustrates the original decision matrix ordered by objective space similarity together with the largest found biclusters and shows the resulting dendrogram. The modules found and the dendrogram help to gain a basic understanding of the problem, even when the decision maker cannot be sure about whether the known solutions are Pareto-optimal or not.

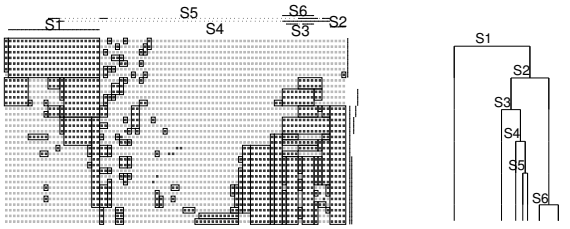

Figure 8: Visualization of structure in a Pareto-set approximation for the network processor design problem: (left) decision space values of the 33 non-dominated solutions found; (right) dendrogram.

For our example instance, 143 out of all 233 decision variables are set to zero for all 33 solutions, which means that certain tasks are never mapped to certain components. Four of the remaining 90 decision variables are set to 1 in all 33 solutions. In this case, it says that in all 33 different processor designs, one particular component, namely a digital signal processor (DSP), is chosen and three of the 25 tasks are allocated to this component. This can assist in decision making in a way that these parts do not have to be taken into account by the decision maker because all known solutions have the same sub-structure. From the dendrogram, we an also extract some information about the problem. For example, in the case of three groups (horizontal cut between S_2 and S_3), one group contains module S_1 (left branch of the dendrogram) and the second one only module S_2 (middle branch). In the third group, indicated by the rightmost branch in the dendrogram of Fig. 8, all solutions contain neither the module S_1 nor the module S_2. S_1 maps all remaining tasks to the DSP. S_2, on the other hand selects a cipher and assigns it two other tasks. Interestingly and similar to the observation for the knapsack problem, all solutions that contain a certain module, here S_1, occur on an extreme of the Pareto-optimal front: the solutions are very cheap but quite slow.

7. CONCLUSIONS

When solving multiobjective optimization problems, three problems occur during decision making: (i) the solutions are represented by too many decision variables, (ii) too many non-dominated solutions exist, and (iii) too many objectives are involved in the evaluation. This study tackled the first and second problem simultaneously by proposing two methods to automatically reduce the number of decision variables and group similar solutions together by finding so-called modules of the decision space, i.e., subsets of decision variables that are as large as possible and are set in as many solutions as possible to the same value. The proposed methods have been extensively tested and compared on random matrices and the 0-1-knapsack problem. In addition, we showed the applicability of these methods in decision making *after* the search in a case study for both the knapsack and a network processor design problem.

In the future, it may be promising to extend the proposed approach to non-binary decision spaces. Here, advanced bi-

clustering techniques could be useful [11]. For a more general approach, modules with arbitrary decision variable values could be considered. Furthermore, one may think of using the reduction techniques online, i.e., *during* the search. The idea would be to reduce the decision space whenever significant modules have been found. Thereby, the search may be better focused towards promising regions.

8. ACKNOWLEDGMENTS

Dimo Brockhoff has been supported by the Swiss National Science Foundation under grant 112079.

9. REFERENCES

[1] S. Bleuler, M. Laumanns, L. Thiele, and E. Zitzler. PISA—A Platform and Programming Language Independent Interface for Search Algorithms. In *Conference on Evolutionary Multi-Criterion Optimization (EMO 2003)*, pages 494–508, Berlin, 2003. Springer.

[2] D. Brockhoff, D. K. Saxena, K. Deb, and E. Zitzler. On Handling a Large Number of Objectives A Posteriori and During Optimization. In *Multi-Objective Problem Solving from Nature: From Concepts to Applications*, pages 377–403. Springer, 2007.

[3] K. Deb and A. Srinivasan. Innovization: Innovating Design Principles through Optimization. In *Genetic and Evolutionary Computation Conference (GECCO 2006)*, pages 1629–1636, 2006.

[4] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1990.

[5] D. E. Goldberg, B. Korb, and K. Deb. Messy Genetic Algorithms: Motivation, Analysis, and First Results. *Complex Systems*, 3:493–530, 1989.

[6] J. A. Hartigan. Direct Clustering of a Data Matrix. *Journal of the American Statistical Association*, 67(337):123–129, 1972.

[7] C. Haubelt, S. Mostaghim, J. Teich, and A. Tyagi. Solving Hierarchical Optimization Problems Using MOEAs. In *Conference on Evolutionary Multi-Criterion Optimization (EMO 2003)*, pages 162–176. Springer, 2003.

[8] J. H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press, 1975.

[9] E. J. Hughes. Radar Waveform Optimization as a Many-Objective Application Benchmark. In *Conference on Evolutionary Multi-Criterion Optimization (EMO 2007)*, pages 700–714, 2007.

[10] M. Křivánek and J. Morávek. NP-hard Problems in Hierarchical-Tree Clustering. *Acta Informatica*, 23(3):311–323, 1986.

[11] S. C. Madeira and A. L. Oliveira. Biclustering Algorithms for Biological Data Analysis: A Survey. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 1(1):24–45, 2004.

[12] S. Obayashi. Pareto Solutions of Multipoint Design of Supersonic Wings Using Evolutionary Algorithms. *Adaptive Computing in Design and Manufacture V*, 2002.

[13] A. Prelić, S. Bleuler, P. Zimmermann, A. Wille, P. Bühlmann, W. Gruissem, L. Hennig, L. Thiele, and E. Zitzler. A Systematic Comparison and Evaluation of Biclustering Methods for Gene Expression Data. *Bioinformatics*, 22(9):1122–1129, 2006.

[14] M. Preuss, B. Naujoks, and G. Rudolph. Pareto Set and EMOA Behavior for Simple Multimodal Multiobjective Functions. In *Parallel Problem Solving From Nature (PPSN IX)*, pages 513–522. Springer, 2006.

[15] K. Sastry, D. E. Goldberg, and X. Llorà. Towards Billion-Bit Optimization via a Parallel Estimation of Distribution Algorithm. In *Genetic and Evolutionary Computation Conference (GECCO 2007)*, pages 577–584. ACM, 2007.

[16] L. Thiele, S. Chakraborty, M. Gries, and S. Künzli. Design Space Exploration of Network Processor Architectures. In *Network Processor Design 2002: Design Principles and Practices*. Morgan Kaufmann, 2002.

[17] D. A. Van Veldhuizen and G. B. Lamont. Multiobjective Optimization with Messy Genetic Algorithms. In *ACM Symposium on Applied Computing*, 2000.

[18] E. Zitzler and S. Künzli. Indicator-Based Selection in Multiobjective Search. In *Conference on Parallel Problem Solving from Nature (PPSN VIII)*, pages 832–842. Springer, 2004.

[19] E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the Strength Pareto Evolutionary Algorithm for Multiobjective Optimization. In *Evolutionary Methods for Design, Optimisation and Control with Application to Industrial Problems (EUROGEN 2001)*, pages 95–100, 2002.

[20] E. Zitzler and L. Thiele. Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach. *IEEE Transactions on Evolutionary Computation*, 3(4):257–271, 1999.

[21] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. Grunert da Fonseca. Performance Assessment of Multiobjective Optimizers: An Analysis and Review. *IEEE Transactions on Evolutionary Computation*, 7(2):117–132, 2003.

APPENDIX

A. OMITTED $\mathcal{NP}$-HARDNESS PROOF

THEOREM 1. *Problem 1 with respect to the error function* $e_{\text{dec}}(\Xi, \Xi^T) := \sum_{1\leq i\leq m}\sum_{1\leq j\leq n}|\xi_{i,j} - \xi^T_{i,j}|$ *is* $\mathcal{NP}$*-hard.*

PROOF. The $\mathcal{NP}$-hardness is shown by a Turing reduction from the $\mathcal{NP}$-hard `SET BASIS` problem [4]. Given a collection $C = \{C_1, \ldots, C_m\}$ of subsets of a finite set S and a positive integer $K \leq |C|$, the decision problem `SET BASIS` is the question of whether there is a collection B of subsets of S with $|B| = K$, such that for each $c \in C$ there exists a subcollection of B whose union is exactly c (page 222 of [4]). The polynomial transformation function we use for the $\mathcal{NP}$-hardness proof is defined as follows: We define a decision matrix $\Xi = (\xi_{i,j})_{|C|\times|S|} \in \mathcal{M}_{|C|,|S|}(\{0,1\})$ such that $\forall 1 \leq i \leq m, 1 \leq j \leq n : \xi_{i,j} = 1 \Leftrightarrow j \in C_i$. With the instance ($\Xi$, K), we ask our oracle for the module set with at most K modules and the smallest error e_{dec}. Then, there exist a collection B of subsets of S with the properties desired in the `SET BASIS` problem if and only if $e_{\text{dec}} = 0$.

Assume there exists such a collection B. This can also serve as a set of modules that yields no error because the following holds. Let us without loss of generality consider the set C_1 and the corresponding subcollection $B' := \{B_1, \ldots, B_p\} \subseteq B$ with $\bigcup_{1\leq i\leq p} B_i = C_1$, i.e., all 1-entries in the first row of Ξ are contained in at least one $B_i \subseteq B'$ such that all decision variables $\xi_{1,j}$ with $j \in B_i$ are set to 1 as well. Thus, $T_{\Upsilon\to\Xi}(T_{\Xi\to\Upsilon}(\Xi))$ contains the same 1-entries as Ξ in the first row and the error is zero.

If $B_1, \ldots, B_K$, on the other hand, are the modules found by our oracle and causing no error, the corresponding subsets of S build an optimal basis for the `SET BASIS` problem: whenever a set $B_1, \ldots, B_K$ of modules causes no error, there is for each 1-entry $\xi_{r,c}$ in Ξ at least one module B_i that contains the column j and where all other columns of B_i are also set to 1 in the rth row of Ξ. Thus, there exists for each row r in Ξ a subset of the $B_1, \ldots, B_K$ such that the union of their columns is the same as the columns that are set to 1 in this row, i.e., the union of the corresponding subsets of S in the `SET BASIS` problem instance is exactly C_i. □