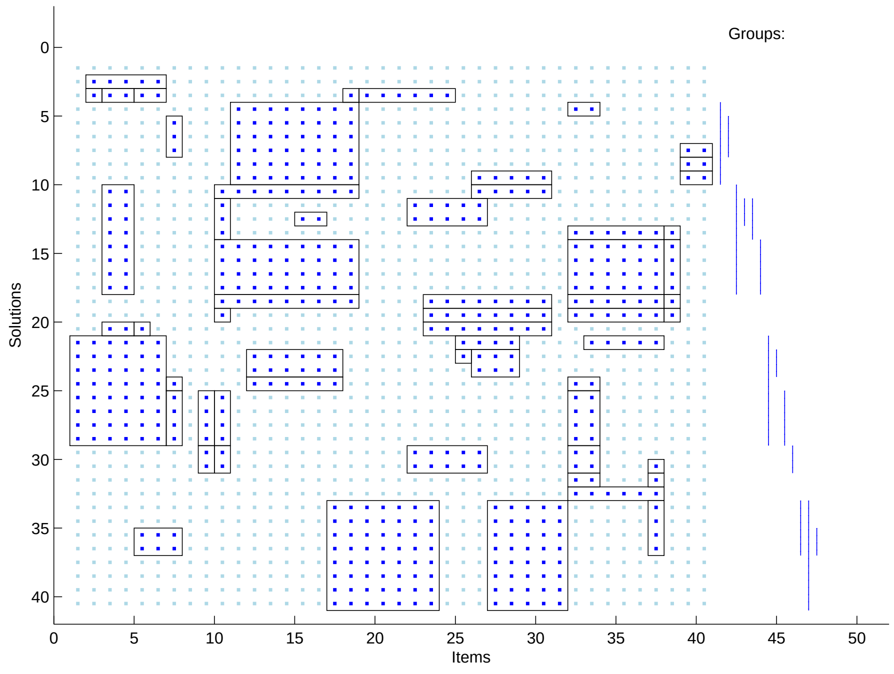
# Pattern Identification in Pareto-Set Approximations

Master Thesis



Tamara Ulrich

Advisor: Dimo Brockhoff
Professor: Eckart Zitzler

ETH Zürich, D-ITET, TIK Computer Engineering and Networks Laboratory

December 18th, 2007

# Contents

# Abstract

When optimizing complex systems, one of the main problems is the size of the search space, as complex problems have many decision variables and require large populations. Two problems emerge. Firstly, reaching the Pareto-optimal front in reasonable time is difficult. Secondly, after an approximation of the Pareto-optimal front has been found, the large number of solutions makes it difficult to decide on a final set of solutions.

This thesis introduces a novel approach to a framework which uses pattern identification to achieve a search space reduction in case of the binary two dimensional knapsack problem. Applied to the decision making problem, it produces hierarchical groups of solutions and helps to learn about the problem structure. When applied to the optimization algorithm itself, grouping of decision variables is used to make the optimization converge faster towards the Pareto-optimal front.

In this thesis it is shown that Pareto-optimal fronts as well as approximations thereof display a high degree of structure. To highlight these structures, several automatic methods to find modules in the binary decision space matrix, and to group solutions with similar modules, are developed. These methods are compared according to multiple test cases and the best one is selected. The groups which are found exhibit a tree structure. As the designer should not be presented with the whole group tree, which in complex systems might become very large, an automatic method to cut the tree on a reasonable level has been developed.

The selected method is then applied to the evolutionary algorithm (EA) at runtime, where the modules are used to reduce the search space during execution of the EA by grouping similar decision variables. In a new reduced representation, these groups of decision variables are varied instead of the individual decision variables. It has been shown that if the modules of the Pareto-optimal front are used, the population converges faster to the optimum than with the original EA. Nevertheless, this behavior could not be observed if modules of the EA population are used.

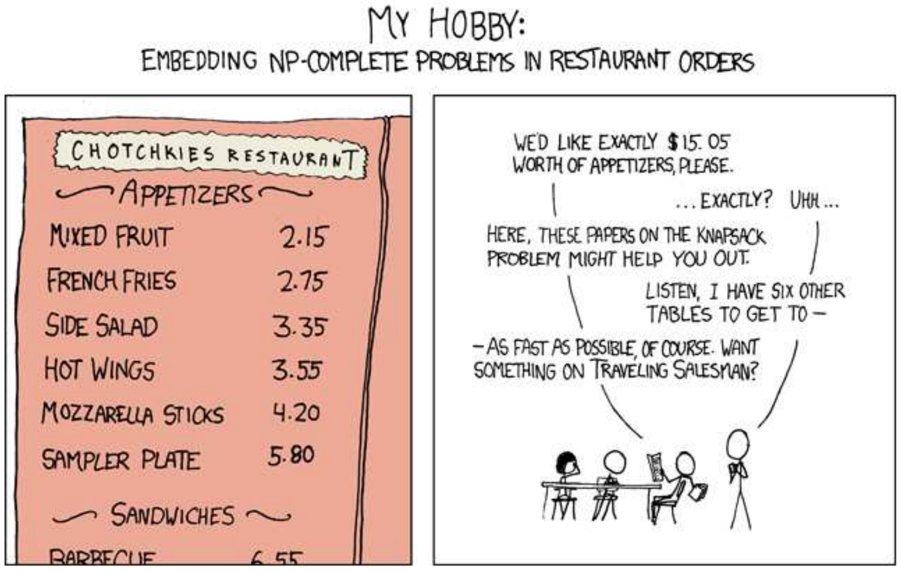
# Acknowledgments

First of all I would like to thank my advisor, Dimo Brockhoff, for his constant assistance during the last six months. He was always there to help if I reached a dead end. Usually simply talking to him solved a great deal of the problem. He always asked the right questions and helped me to get to the bottom of various problems. I would also like to thank him for his patience when revising this work and the corresponding presentation.

Furthermore I would like to thank Prof. Dr. Eckart Zitzler for his ongoing support. Having spent hours and hours in his office, discussing problems and promising approaches on how to solve them, provided me with an excellent insight into what life as a researcher might be like. His continuous devotion to this project always gave me the impression of being a full member of the team.

# Chapter 1

# Introduction


MY HOBBY:
EMBEDDING NP-COMPLETE PROBLEMS IN RESTAURANT ORDERS
CHOTCHKIES RESTAURANT
APPETIZERS
MIXED FRUIT 2.15
FRENCH FRIES 2.75
SIDE SALAD 3.35
HOT WINGS 3.55
MOZZARELLA STICKS 4.20
SAMPLER PLATE 5.80
SANDWICHES
WE'D LIKE EXACTLY $15.05 WORTH OF APPETIZERS, PLEASE.
...EXACTLY? UHH...
HERE, THESE PAPERS ON THE KNAPSACK PROBLEM MIGHT HELP YOU OUT.
LISTEN, I HAVE SIX OTHER TABLES TO GET TO –
– AS FAST AS POSSIBLE, OF COURSE. WANT SOMETHING ON TRAVELING SALESMAN?

[1]

## 1.1 Motivation

When modeling complex systems, a search space exploration may be necessary, where all solutions are compared with respect to the different objectives to find an optimal solution. Usually it is not possible to optimize all objectives simultaneously, as most objectives are conflictive.[2] Because there is not a single optimal solution, the exploration results in a set of incomparable solutions, called the Pareto-optimal set.

[1]Taken from xkcd: http://xkcd.com

[2]Example: faster systems usually are more expensive than slower ones

As an exact calculation of the Pareto-optimal set usually is not feasible, an approximation of the Pareto-optimal set is calculated, for example using evolutionary algorithms.

With increasing number of objectives, the Pareto-optimal set contains more and more incomparable solutions, which makes larger populations necessary. The larger populations in turn lead to two problems. Firstly, it takes longer to reach the Pareto-optimal front with larger populations, and secondly, it becomes more difficult to decide on a final set of solutions from the approximation set.

It is the goal of this thesis to find structures in a given set of solutions and to reduce it with this information. The reduction of a population consists of two steps. Firstly, groups of decision variables that occur in several solutions are merged into modules. Secondly, solutions with similar modules are combined into groups.

The decision space of a set of solutions is represented by a matrix. In this matrix, each row represents a solution, whereas the columns are the decision variables. For the sake of simplicity, only binary decision spaces are considered in this thesis.

In other words, it is the goal of this thesis to reduce a binary matrix in both dimensions. The width of the matrix may be reduced by finding modules, whereas the height of the matrix is reduced by grouping rows with similar modules.

Two scenarios, as described in the following sections, are studied in this thesis.

### 1.1.1 Offline Scenario

In the offline scenario, the optimization has already been completed, resulting in a Pareto-optimal front or an approximation thereof. Using an appropriate algorithm which has yet to be chosen, modules have to be identified. Afterwards solutions containing similar modules are grouped, also taking into account the respective objective space values of the solutions.

### 1.1.2 Online Scenario

In the online scenario, the algorithms developed for the offline scenarios will be adapted for the use during optimization. This adaption aims at reaching the Pareto-optimal front faster than with the original evolutionary algorithm by reducing the search space. To this end, the focus primarily lies on groups of decision variables, not on groups of solutions like in the offline scenario.

## 1.2 Related Work

### 1.2.1 Offline Scenario

In this thesis, the goal of the offline scenario is to find structure in solution sets, to reduce the search space with this information and to group solutions to learn about the problem at hand.

Several projects exist which provide reasons to believe that there actually is a certain degree of structure in Pareto-optimal sets or approximations thereof.

Preuss et al. [1] examined the connection between decision and objective space. For two parameters and two objective space values, they constructed a sample problem where a well structured distribution of solutions in decision space maps on a less structured solution distribution in objective space. They showed that an investigation of the objective space alone in some cases does not suffice to reach good solutions.

Deb and Srinivasan [2] introduced a new design methodology called *Innovization*, which uses optimization methods not primarily to find optima, but to develop new design principles by gaining a deeper understanding of the problem at hand. Considering among others a brake design example with two conflicting objectives, they found that two out of five decision variables have the same value for all solutions in the Pareto-set approximation. With this result they showed that not all decision variables are necessary to represent the system.

Hartigan [10] developed a direct clustering algorithm which produces a partitioning of the matrix. It partitions the matrix into biclusters. Biclusters are defined as a set of columns which behaves similarly across a set of rows. These biclusters exhibit a tree structure in both axes, which means that when these biclusters are mapped to the axes, two biclusters either have to be disjoint or one has to include the other. A newer biclustering algorithm called Bimax [11] finds an exhaustive set of all biclusters of ones in a binary matrix. Both of these algorithms are used in this thesis to automatically find biclusters in a decision space matrix.

In Biology, clustering algorithms are usually applied to find groups in gene expression data. These algorithms, for example clustering based on Euclidean distance, group the genes according to some distance measure. The problem with this approach is that the similarity between genes are not based on similarities in certain groups of conditions, but on similarities over all conditions, which may lead to the masking of smaller, less prominent groups of conditions. This problem can be solved with the concept of biclustering, where genes and conditions are grouped simultaneously. Cheng [3] introduced a node-deletion algorithm which finds biclusters in gene expression data that have low mean squared residue scores. These biclusters are then used to gain information about gene groups and about classes of conditions.

Other standard techniques exist which aim at dimensionality reduction. Principal Component Analysis for example reduces the dimensionality by transforming the data to a new coordinate system. Feature extraction, on the other hand, reduces dimensionality by minimizing redundancy in describing data. It strives to find a set of features which is easily and fast to describe but which still represents the data accurately.

Most algorithms encounter problems when being run on complex systems. As the number of decision variables and solutions grows, calculations of complex algorithms become infeasible. While other algorithms like feature selection work well on large data sets, similarities in the data set might be lost. This is why in this thesis, the idea of search space reduction, which makes the handling of large data sets feasible, is combined with biclustering and grouping of solutions, which preserves the initial structure of the data set.

### 1.2.2 Online Scenario

Other approaches try to speed up the optimization by varying blocks of decision variables instead of varying the decision variables individually. Such blocks of decision variables may be defined as schemata, which were introduced by Holland [4, 5]. A schema is a similarity template which is of the same length as each solution and contains fixed numbers and don't care characters. The fixed numbers indicate bit positions that are fixed in the respective schema, and don't cares stand for those positions which are not fixed. As an example, the schema *1*0 matches the string set {0100, 0110, 1100, 1110}.

The most prominent example using this schemata is Goldberg's messy GA (mGA) [8, 9], which manipulates special schemata. It is based on the schema theorem that has been postulated by Goldberg [6]:

> "Short, low-order, above-average schemata receive exponentially increasing trials in subsequent generations."

Short in this case means that the distance between the first and the last specified position is small. Low-order schemata are schemata with only few specified bit positions. Schemata which are both short and low-order are less likely to be disrupted by crossover. Above-average schemata match solutions which have high fitness values and therefore are more often selected. These short, low-order, above-average schemata are called building blocks[3].

The special schemata which Goldberg's messy GA manipulates are these building blocks. In the mGA, all possible building blocks of a specified size are generated at the start of the algorithm. Because this enumeration of building blocks is computationally expensive, the fast messy GA (fmGA) was introduced, where the building blocks are initialized probabilistically.

Haubelt et al. [15] have introduced a hierarchical problem description of an initially non-hierarchical problem to reach the Pareto-optimal front faster. They proposed an adaption of mutation and recombination methods to deal with the hierarchical decision vectors.

The problem with both the Goldberg and the Haubelt approach is that the representation reduction has to take place before the optimization. And at least in the Haubelt approach, the user needs to have some knowledge about the system structure to be able to generate the hierarchical description of the problem.

Therefore in this thesis, a new approach is devised, which makes it possible to do the representation reduction online and automatically. It is also possible to switch back and forth between the reduced and the ordinary representation to optimize the speed of the optimization.

[3]See [7], Chapter 4.2 for a more detailed building block overview

# Chapter 2

# Offline Scenario

In this chapter, the approach to the offline scenario is delineated. After defining the offline problem, several ideas are discussed that have been discarded in the end. Then, the choice of biclustering algorithms, which are used to find modules, is illustrated. The two chosen algorithms and their implementation is described in the next sections. Finally, an automatic method is developed which splits the generated group tree at a reasonable level.

## 2.1 Goal

When optimizing complex systems, several difficulties emerge both during the optimization and after the search, when the Pareto-set approximation is given. These difficulties are due to the fact that in complex systems, there are many decision variables and therefore the search spaces become very large. Large search spaces in turn lead to many Pareto-optimal solutions.

The main problem which the offline scenario deals with is that in decision making, after the search, it is difficult to decide on a final set of solutions out of the many solutions of the Pareto-set approximation.

To tackle these problems, a framework has to be developed which provides the means to automatically identify patterns in the Pareto-set approximations. These patterns are then used to achieve a search space reduction.

### 2.1.1 Definition of Offline Problem

The reduction is composed of two steps. In the first step, modules are automatically identified in the decision space representation of the Pareto-set approximation. These modules highlight similarities between decision space variables. Modules are defined as biclusters of ones. This is due to the fact that in selection problems, such as the knapsack problem, it is more interesting to know which items are selected in a solution than to know which items are not. Furthermore, there are codings as for example for the network processor design of the EXPO tool [16], where there are much more zeros than ones, and therefore the main focus lies on the ones.

The goal is to make these biclusters of ones, or modules, as large as possible. A bicluster is defined as a set of columns which behaves similarly across a set of rows. To find these modules automatically, two biclustering algorithms are considered.

In the second step, these modules are used to group solutions. To this end, solutions which have similar modules are merged into groups. For an easier representation of groups and to make the selection of groups more simple, a tree structure of the groups is required, which means that when mapped to the axes, two biclusters either have to be disjoint or one has to include the other. In cases where the groups are not directly given by the modules, grouping methods have to be developed which assure the tree structure of the groups.

The goal of the groups is to contain solutions which are as similar as possible. Similarity has to be defined both in the decision and the objective space. As the designer in the end should not be presented with the whole group tree, the tree has to be cut on a certain level. A method has to be developed which automatically generates this best level cut.

For the sake of simplicity, only binary decision spaces are considered. Starting with two objectives, the methods need to be developed such that they may also be applied to higher dimensional objective spaces. As a test problem, the two dimensional knapsack problem is chosen, where the decision variables are called items.

The decision space representation of a Pareto-set approximation is given in a binary matrix, where the rows are the solutions and the columns are the decision variables. A one indicates that the corresponding decision variable is selected in the respective solution.

In a first step, the degree of structure in Pareto-set approximations has to be identified. This is done in the following sections.

## 2.2 Discarded Ideas

Before starting on finding modules and groups, several other approaches were considered to reduce the search space, both in the decision variable direction and in the solutions direction. As these ideas influenced the choice of the final approach, they are briefly discussed in this section.

### 2.2.1 Coding Similar Columns Blocks

The straight forward way to reduce decision variables is to code similar blocks of decision variables. In this approach, groups of decision variables which only occur in a subset of configurations are represented by the necessary number of bits. If, for example out of three decision variables all solutions either only contain the first decision variable, or otherwise none of these three decision variables, then these decision variables may be represented by one bit.

The first problem that emerged was the lack of an appropriate quality measure of the coding. Minimizing the number of coding bits was considered, but in this case it is obvious that the best result will be achieved by coding the $n$ different decision variables by $\log_2 n$ bits. But with such a coding, all information about similarities between decision variables and between solutions will be lost, and different solutions will be assigned a random new representation number which does not depend on the initial representation. Such a reduction is of no use if the solutions shall still be grouped after the reduction.

The second problem was the fact that blocks of decision variables can be of any size. Therefore testing all possible blocks is infeasible for large search spaces.

### 2.2.2 Discarding Identical and Complementary Columns

Depending on the structure of the decision space, it is possible that one decision variable can be found in exactly the same solutions as another decision variable. In this case, the two decision variables may be combined into one decision variable, as they are identical. The same goes for pairs of decision variables if all solutions contain exactly one of the two decision variables. In this case, the decision variables are said to be complementary.

Tests showed that there usually are less than one percent similar or complementary decision variables (see Table 2.1, where for each matrix size, the average was taken over 11 different problems). In this table, the absolute numbers indicate the number of columns which are either identical or complementary to another column. The percentage values indicates the percentage of columns that may be discarded in each case. Columns which contain only ones or only zeros have been discarded prior to these tests.

| | | Number of decision variables | | | |
|---|---|---|---|---|---|
| | | 100 | | 300 | |
| | | Identical | Complementary | Identical | Complementary |
| Population size | 100 | 1.0% (1.0) | 1.6% (1.6) | 3.5% (10.5) | 5.3% (15.9) |
| | 300 | 0.3% (0.3) | 0.5% (0.5) | 1.4% (4.1) | 1.8% (5.5) |
| Pareto Front | | 0.1% (0.1) | 0.1% (0.1) | 0.1% (0.4) | 0.1% (0.2) |

Table 2.1: Number of Identical and Complementary Decision Variables

### 2.2.3 Iterative Biclustering with Bimax

In this approach, all biclusters of ones are found in the decision space matrix of the problem. Then, a subset of all biclusters is selected. The subset is selected such that it covers as many ones of the matrix with as few biclusters as possible.

Afterwards, a new representation is generated, in which the new decision variables are the selected biclusters. So each solution is now defined by the set of biclusters which it contains.

The Bimax algorithm can then be applied to this new representation again and again. After each Bimax run, solutions with similar representations are grouped. This becomes possible because when the selection of biclusters is made, an error is usually allowed to assure that the new reduced representation does not have more decision variables than the old one.

In the original representation, declaring a few ones as error does not lead to any problems. In the reduced representation though, where each one represents an arbitrarily sized set of decision variables, declaring a one as error may distort the representation of the affected solutions considerably.

In the test runs, it has been found that most solutions are grouped because they do not contain any biclusters anymore, as all ones in the representations have finally been declared as error. This is of no use since the groups should be generated based on the similarities between the solutions.

## 2.3 Selection of Biclustering Algorithms

Since the approaches proposed in the above section exhibit major drawbacks, the initial idea of identifying modules in the decision space matrix of the problem is considered next.

To automatically find modules in the matrices, a biclustering algorithm has to be selected. [12] gives a good overview over different bicluster types and the appropriate methods to find them. In the case of a binary matrix, the main interest lies on biclusters with constant values, specifically biclusters which only contain ones or which only contain zeros.

Hartigan [10] proposed one of the first algorithms to find blocks of constant values by finding a partitioning of the matrix. Several other algorithms are based on his ideas, for example [13], which extends the original Hartigan algorithm, or [14], which follows up on the divide-and-conquer strategy introduced by Hartigan. The Hartigan algorithm has many advantages. It is exhaustive, which means that all elements of the matrix belong to at least one bicluster. Its implementation is simple and its computation fast, due to the divide-and-conquer strategy. The resulting biclusters are highly structured and a tree structure can be required on both axes. The drawbacks are that the biclusters cannot overlap and can only encompass neighboring solutions.

Therefore, as a counterpoint to the Hartigan algorithm, the Bimax algorithm [11] has been chosen. While the Hartigan algorithm is simple and well structured, the Bimax algorithm finds an exhaustive set of biclusters containing only ones. The advantage of such an approach is that the resulting biclusters are very flexible, as there is no restriction of the size or shape of the biclusters, and they are also allowed to overlap. The drawback is that these biclusters do not structure the matrix at all and groups are not implicitly given. Therefore, several grouping methods have to be developed to find an appropriate grouping based on the given biclusters.

These two biclustering algorithms will be explained in detail in the following two sections.

## 2.4 The Hartigan Algorithm

The biclustering algorithm introduced by Hartigan [10] is a direct clustering algorithm, also known as *Block Clustering*, which is based on the divide-and-conquer principle. This algorithm produces a partitioning of the matrix into biclusters of ones or zeros. The resulting biclusters of ones are the modules of the matrix, and the tree structure which may be required on either one or both axes defines the grouping of the solutions. A tree structure means that if the biclusters are mapped on the axis, they either have to be disjoint or one bicluster needs to include the other. This ensures that the matrix can be split along the borders of the largest bicluster without disrupting any smaller biclusters.

The pseudocode of the Hartigan algorithm looks as follows:

```
Sort the matrix

while splittable /* there still are biclusters which contain both ones
and zeroes and the minimum bicluster size is not yet reached */
  select the next best splitt /* according to chosen split measure */
  update fixed splits /* see Figure 2.2 */
  calculate and store the next best split of the two new submatrices
end while
```

The algorithm starts by sorting the matrix in both axis directions. It then proceeds to split the matrix into two sub-matrices, followed by iteratively splitting the created sub-matrices until each sub-matrix either reaches a minimum size or contains only ones or only zeros. If it reaches the minimum size without containing only ones or only zeros, the ones in the sub-matrix are declared as error.

To define the characteristics of the algorithm, three decisions have to be made. The first one is which method should be used to sort the matrix prior to the biclustering. The second one is which measure should be used for the splitting in the divide-and-conquer strategy. And the last one is whether the tree structure which is achieved by making use of fixed splits is only required on one, on both or on none of the axes.

The methods which were developed for these three decisions are explained in the following sections.

### 2.4.1 Presorting of Matrix

As the algorithm produces a partitioning of the matrix using a divide-and-conquer strategy, the row and column order cannot change during algorithm execution. It is therefore important to find an appropriate method to sort the matrix prior to the bicluster calculation. Several methods that have been implemented are described in the following subsections.

#### Sorting According to Row Sums and Columns Sums

This is the sorting method proposed by Hartigan. It is designed to be applied to a matrix consisting of real values. When applied to a binary matrix, several problems arise. The biggest drawback is that in a binary matrix, the probability of several rows or columns having the same sum value is quite high. The sum value yields no information about the location of the ones. An example to demonstrate this problem can be found in the following matrix:

$$\begin{pmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix} \Longrightarrow \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix}$$

Although this matrix could be easily regrouped as shown above, it is not when sorting according to row and column sums, as row sums are equal to 2 for every row and column sums are equal to 4 for every column.

### Sorting According to Distance Measure

This method divides the vectors into two groups. All vectors are iteratively assigned to these two groups. First of all, the two vectors that are furthest from each other are calculated, considering the chosen distance measure. These two vectors provide the foundation of the two groups and are placed at the opposing edges of the sorted matrix. The method then iteratively selects from the remaining vectors the vector which is nearest to one of the last two vectors that have been added to the groups and places it next to this last selected vector, towards the middle of the matrix.

The following distance measures are considered, where a higher value means a higher distance:

- Hamming distance: The percentage of bits for which the values of the two vectors differ.
- Jaccard distance: One minus the number of bit pairs that are both one divided by the number of bit pairs that have at least 1 one. $1 - \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cup B| - |A \cap B|}{|A \cup B|}$
- City Block distance: In the binary case the same as the Hamming distance.
- Minkowski distance: Includes the City Block (1-norm Minkowski distance) and the Euclidean distance (2-norm Minkowski distance).
- Cosine distance: $\cos(x, y) = 1 - \frac{<x,y>}{||x|| \cdot ||y||}$
- Chebychev distance: Maximum distance of all pairwise distances, which in the binary case is either zero, if the vectors are the same, or one, if the vectors differ.
- Entropy: The entropy is calculated according to the formula $H(X|Y) = \sum_{(Y=y)} P(Y = y) H(X|Y = y)$, where $H(X|Y = y) = \sum_{(X=x)} P(X = x|Y = y) \text{ld} P(X = x|Y = y)$. The problem with pairwise entropy is the same as with the Chebychev distance. It is either zero for identical vectors or one for different vectors.

To compare these distance measures, an exhaustive set of vector pairs of length 4 is considered:

$$\begin{pmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 1 & 0 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \end{pmatrix}$$

While the pairs in the first row are identical, those in the second/third/fourth/fifth row vary by one/two/three/four bits.

Based on these vector pairs, the cosine distance measure may be discarded, as it produces divisions by zero. The other distance measures are shown in Figure 2.1.

It can be seen that the distance is zero for all measures as long as the two vectors are identical (vector pairs 1 to 5, order corresponding to above enumeration). The Chebychev and the Entropy measures may be discarded as well, as they switch to distance one as soon as a variation is introduced. The Hamming, City Block and Euclidean distances behave similarly, except for a factor. The Hamming distance is chosen to represent this kind of distance measure.

Therefore, the only two distance measures that remain are the Hamming and the Jaccard measure.

### Sorting According to Objective Space Value

The underlying assumption of this method is that solutions which are close in objective space have similar decision space representations. This method assures that the calculated groups are always close in objective space, as partitions in the matrix can only encompass neighboring solutions.

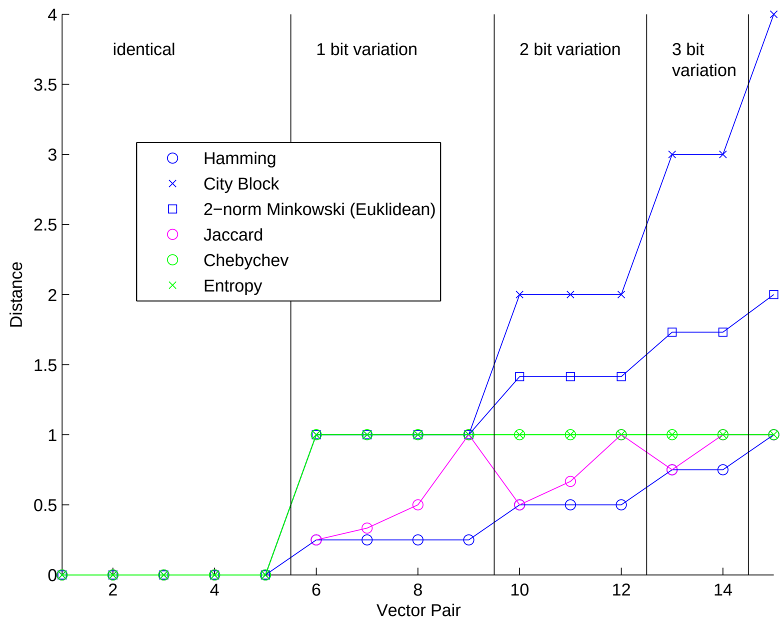


Figure 2.1: Distance Measures Comparison

Sorting according to objective space values is straightforward for two dimensional objective spaces, as Pareto-optimal fronts or approximations thereof consist of non dominated individuals. Hence, if one individual is better in one objective, it supposedly is worse in the other objective.

In case of a three or higher dimensional objective space, additional rules have to be developed. One method is discussed in Section 2.4.3 on derived algorithms.

### 2.4.2 Split Measure

After sorting, the split measure is used to iteratively split the matrix into partitions, which will finally lead to a set of biclusters. In this thesis, three split measures have been considered. The first one is the measure proposed by Hartigan. But as this measure relies on the sorting of the matrix by row and column sums, a more general measure which is based on the percentage of ones was devised in this thesis. In addition to this basic measure, a more elaborate measure based on entropy was implemented.

#### Hartigan Measure

In accordance with [10], the Hartigan measure is defined by the sum of squares reduction (SSQR). If, as an example, the matrix $B_p$ has to be split into submatrices $B_p'$ and $B_p''$ the following formula is used:

$$\text{SSQR} = c_p r'_p (A(B'_p) - A(B_p))^2 + c_p r''_p (A(B''_p) - A(B_p))$$

where $A(B)$ denotes the average of $A$ over the block $B$. The best split maximizes the sum of squares reduction. Hartigan has shown that the best split will occur in order of the row means, i.e. all row means above the split will be less than the row means below the split. Hence, it suffices to test only splits between subsequent rows if the matrix is sorted according to row sums.

One problem faced by this measure is that even if the matrix is sorted according to row sums, some sub-matrices where splits have to be found may be not. Consider the following matrix:

$$\begin{array}{|ccc|cccc|}
\hline
0 & 0 & 1 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 \\
\hline
\end{array}$$

Although the matrix as a whole is sorted correctly according to row sums (3 and 4) and column sums (0, 0, 1, 1, 1, 2 and 2), the left sub-matrix is not, as its upper row has a higher row sum than its lower row.

### Percentage Measure

The basic measure proposed in this thesis is defined as the difference of percentage of ones in the two sub-matrices created by the split, where the best split maximizes the following function:

$$\left\| \frac{\text{sum}(B'_p)}{\text{size}(B'_p)} - \frac{\text{sum}(B''_p)}{\text{size}(B''_p)} \right\|$$

Similar to the Hartigan measure, the percentage measure is only computed for splits between subsequent rows. The best split may or may not be one of these computed splits, but as the matrix cannot be resorted during algorithm execution, only these splits are available.

### Entropy Measure

This measure strives to minimize the sum of joint entropies of the two sub-matrices created by the split. For a column split, the joint entropy of the rows of each sub-matrix is calculated. For a row split on the other hand, the joint entropy of the columns is calculated.

The problem of the Entropy split measure is the higher calculation complexity. While both the Hartigan and the Percentage measure were only dependent on the number of rows/columns for a row/column split, the Entropy measure depends on both row and column number for each split.

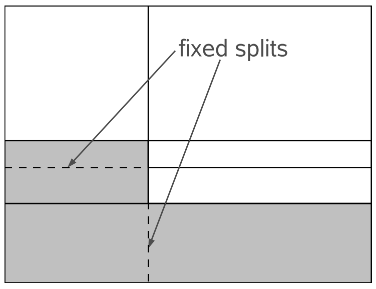


Figure 2.2: Fixed Row and Column Split

### 2.4.3 Derived Algorithms

#### Relaxation of Tree Structure Restrictions

In the original algorithm proposed by Hartigan, a tree structure on each axis is required, i.e. two biclusters either have to be disjoint, or one must include the other on that particular axis. The advantage of such a tree structure is that it is easier to represent and group the solutions and decision variables. However, if there is no need for easy representation or grouping, the tree restriction should be relaxed, as it limits the number of possible splits as shown for example in Figure 2.2, where the upper gray box does only have one possible row split, and the lower gray box only has one possible column split.

When the biclusters, as in the offline scenario, will later be used to group solutions or rows, a tree structures is only required on the y-axis. For the online scenario, where decision variables will be grouped, a tree structure on the x-axis is required.

#### Relaxation of Sorting Restriction

The algorithm proposed by Hartigan can be adapted such that sorting of the sub-matrices is allowed before each split. The goal of this adaption is to find better splits. The drawback is the loss of the tree structure, which clearly defines the solution groups.

Without the tree structure, the problem is similar to the Bimax problem after the biclusters have been found. Therefore, the grouping methods developed for the Bimax biclusters (see Section 2.5.3) are applied to this adapted Hartigan problem.

#### Hartigan for Higher Dimensional Objective Spaces

Up to now, only two dimensional objective spaces have been considered. Two methods depend on the objective space values. The first one is the presorting method of the decision space matrix. The second one is the definition of the importance or largeness of a module[1], which may be defined in various ways,

[1]See Section 2.5.3 on the definition of largeness of modules

like the number of ones it contains in the decision space, or the deviation which is caused in the objective space if this module is discarded.

The deviation of the objective space values has been defined as the sum of Euclidean distances between each point before and after the module has been discarded. The definition of the Euclidean distance in general is defined for n-dimensional spaces. Therefore, this poses no problems.

For the presorting of the decision space matrix, the adaption is not so easy. In contrast to the two dimensional case, in which the two objective space values of indifferent solutions are inversely proportional, it is not clear anymore how to sort the matrix according to objective space values in higher dimensional spaces, as sorting the individuals yields different sorting orders, depending on the dimension in which the sorting takes place.

One possible solution is to sort the matrix according to all dimensions, and compare the split values of the best splits, where the split with the highest value will be selected. In case of the derived Hartigan algorithm with free sorting before each split, this procedure may be repeated before each split.

## 2.5 The Bimax Algorithm

Bimax is a fast divide-and-conquer algorithm proposed in [11]. In a binary matrix, it finds every bicluster of ones, which of course can overlap with other biclusters.

### 2.5.1 Finding Biclusters

In contrast to the Hartigan algorithm, there is only one way to find the biclusters with Bimax. With bigger matrices, finding all biclusters is infeasible. In this case the minimum bicluster size may be increased, thereby introducing an error, as there might be some ones that are only contained in smaller biclusters.

### 2.5.2 Selecting Modules

As most biclusters overlap, a subset of biclusters shall be chosen that will later be used for grouping. The goal is to cover as many ones with as few biclusters as possible. As testing all bicluster configurations is infeasible, a deterministic method has to be devised. The method proposed in this thesis is straightforward. It iteratively selects the bicluster which covers most of the remaining ones that are not yet contained in any previously chosen bicluster. These selected biclusters are then defined as the modules.

### 2.5.3 Grouping of Solutions

In contrast to the algorithm proposed by Hartigan, the modules found with Bimax have no tree structure on the y-axis. Therefore, a new way to group the solutions according to randomly distributed modules has to be found. In this thesis, the solutions are grouped by applying one of the following two methods.

### Increasing Group Method

The increasing group method starts with the largest module, where the definition of largeness has yet to be decided on. This module divides the solutions into the first group and the rest, which has not been grouped yet. In order to maintain a tree structure on the y-axis, all other modules are split according to this first modules boundaries. Then, the next largest module is selected, yielding the next group. This procedure continues until no modules with more than one row remain.

### Decreasing Group Method

This method subsequently discards modules, starting with the smallest modules. The ones contained only in these discarded modules are considered as error. As soon as two solutions have all their different bits considered as error, they are merged into a group.

### Comparison

Both methods have their drawbacks. The increasing group method has to deal with the problem that after each step, the remaining modules are split along the already existing group boundaries. One way to alleviate this problem is to adapt the definition of a large module to benefit modules that contain more rows. For example a higher weight could be assigned to the number of rows (solutions) that a module contains.

The decreasing group method, on the other hand, has the problem that solutions with a high error are more likely to be grouped with other solutions than solutions with a low error, even if there are only a few similarities with other solutions in the group.

As an example to illustrate both methods, the following matrix is considered. In this example, the largeness of a module is defined by the number of ones it encompasses.

The increasing group method first selects the module with 8 elements, thereby grouping solutions 2, 3, 4 and 5 and splitting the modules with 6 and 4 elements into a 4-element and a 2-element and into two 2-element modules, respectively. Then it selects the remaining 4-element module from the initial 6-element module, thereby grouping solutions 4 and 5. As none of the remaining modules encompasses more than one solution, grouping stops. (Note: $X$ stands for the ones of selected modules).

$$\begin{array}{ccccccc} 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ X & X & 0 & 1 & 0 & 1 & 1 \\ X & X & 0 & 0 & 0 & 0 & 0 \\ X & X & 0 & 1 & 1 & 0 & 0 \\ X & X & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \end{array} \Rightarrow \begin{array}{ccccccc} 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ X & X & 0 & 1 & 0 & \overline{1} & \overline{1} \\ X & X & 0 & 0 & 0 & 0 & 0 \\ X & X & 0 & X & X & 0 & 0 \\ X & X & 0 & X & X & 0 & 0 \\ 0 & 0 & 0 & \overline{1} & \overline{1} & 0 & 0 \end{array}$$

The decreasing group method, on the other hand, first groups solutions 4 and 5, as they are identical. Then it discards the module with one element, afterwards the module with 4 elements. Now it is able to group solutions 2 and 3. Finally it discards the module with 6 elements and groups solutions 2, 3, 4 and 5. (Note: $X$ stands for the ones of discarded modules).

$$\begin{array}{ccccccc} 0&0&0&0&0&1&1\\ 1&1&0&X&0&1&1\\ 1&1&0&0&0&0&0\\ 1&1&0&1&1&0&0\\ 1&1&0&1&1&0&0\\ 0&0&0&1&1&0&0 \end{array} \Rightarrow \begin{array}{ccccccc} 0&0&0&0&0&X&X\\ 1&1&0&0&0&X&X\\ 1&1&0&0&0&0&0\\ 1&1&0&1&1&0&0\\ 1&1&0&1&1&0&0\\ 0&0&0&1&1&0&0 \end{array}$$

$$\Rightarrow \begin{array}{ccccccc} 0&0&0&0&0&0&0\\ 1&1&0&0&0&0&0\\ 1&1&0&0&0&0&0\\ 1&1&0&X&X&0&0\\ 1&1&0&X&X&0&0\\ 0&0&0&X&X&0&0 \end{array} \Rightarrow \begin{array}{ccccccc} 0&0&0&0&0&0&0\\ 1&1&0&0&0&0&0\\ 1&1&0&0&0&0&0\\ 1&1&0&0&0&0&0\\ 1&1&0&0&0&0&0\\ 0&0&0&0&0&0&0 \end{array}$$

In this example, the resulting groups of the two methods are quite similar, given that they use the same measure to define the largeness of modules. Both methods group solutions 2, 3, 4 and 5 as well as solutions 4 an 5. The decreasing group method even produces an additional group. In this case, grouping solutions 2 and 3 is not a good choice, because their only similarity is the 8-element module, which is also producing the group of solutions 2, 3, 4 and 5. There is no reason to group solutions 2 and 3 separately.

### Definition of Large and Small Modules

The definition of the largeness of a module indicates the order in which modules are selected or discarded in the increasing and decreasing group method. In this thesis, several definitions are considered:

- Size in Decision Space: The simplest definition of largeness is the number of rows times the number of columns of a module.
- Free Ones in Decision Space: This definition yields the order used for module selection. It is always calculated from largest to smallest module and is defined as the number of ones in the module, which are not yet contained in any previously chosen module.
- Fixed Free Ones in Decision Space: With this definition, the largeness of a module is the number of ones that are exclusively contained in this module. The advantage of this definition over the Free Ones definition is that it does not depend on the order of selection. The drawback is that it depends on all modules, not only the selected modules.
- Distance in Objective Space: Large modules should also be close in objective space, as they define the grouping. Therefore, a definition that takes

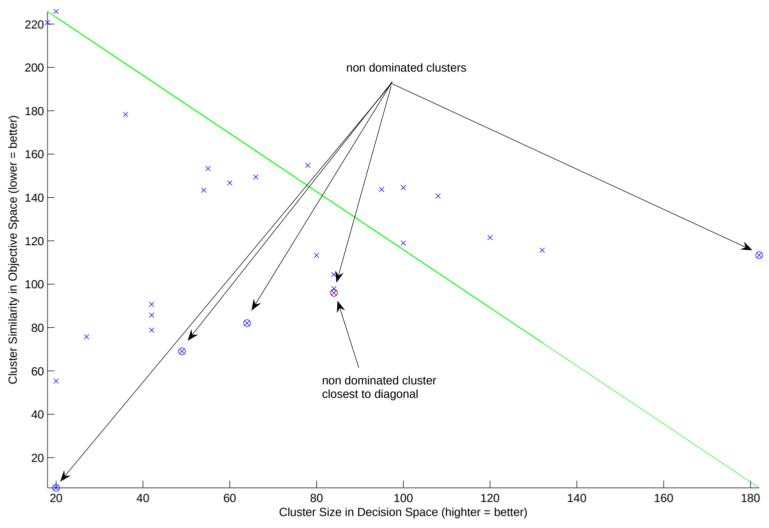


Figure 2.3: Example Module Selection

the objective space into account is considered. For example the average pairwise Euclidean distances of the objective space representation of the solutions of this particular module could be calculated. This value can then be weighted with the module size in the decision space.

- Deviation in Objective Space: The largeness of the module is defined as the deviation in objective space that is caused by discarding the module. As a distance measure, the sum of the Euclidean distance differences of all solutions can be considered.

- Selection based on Rank, Size and Similarity in Objective Space: The problem with definitions based only on the objective space is that the closeness of the solutions of a module in the objective space depends on the number of solutions in the module. Modules with fewer solutions are usually closer in object space than modules with more solutions.

  Therefore, a more sophisticated method to order the modules has been developed. It selects modules based on a plot where the x-axis is the size in decision space and the y-axis is the similarity in objective space. The goal is to find the modules which are reasonably large, but still close in objective space.

  In a first step, a straight line from to the worst point of the plot to the best point is drawn. For selecting the best module, only modules not dominated in the plot by remaining modules are considered in each step. The best module is the one closest to the line. An example of such a module selection is shown in Figure 2.3, where the problem has 50 items with seed 1.

## 2.6 Best Level Cut of Group Tree

Until now the grouping has led to a tree structure that contains every group that has been found. This tree becomes quite large, the more complex the system is and the more decision variables and the more solutions there are in the Pareto-set approximation. In the end, the user should only be presented with a part of this tree, thereby reducing the tree to a reasonable number of groups. The goal is to find a point where the error is smallest and the similarities of the remaining groups is largest. To find this best level cut, the modules and the ones they contain as well as groups and their similarities are plotted.

### 2.6.1 Structure/Error Plot

This plot shows how the number of modules corresponds to the ones contained in chosen modules. The error is defined as the number of ones that are not yet contained in an already chosen module. Modules are selected from largest to smallest. When only the largest module is selected, the number of contained ones is smallest and the error is largest. Given a set of already chosen modules, the next best module in this case is the module that covers most of the ones that are not contained in one of the already chosen modules. In each step, the next best module is selected, lowering the error gradually by increasing the ones contained in modules. Plotting the number of already chosen modules against the inverse error (i.e. the number of ones contained in the chosen module) yields the structure/error plot.

An example of such a structure/error Plot may be seen in Figure 2.4. Here, the error is defined as ones not yet covered by already selected modules. In this figure, the cumulated inverse error, which is similar to the cumulated number of ones covered by the modules, is plotted.

### 2.6.2 Group/Similarity Plot

For each group, the similarity is calculated, either in the decision or the objective space. The groups are then sorted according to their similarity, best first, and plotted against their cumulated similarity, which yields the group/similarity plot.

There are several problems concerning this kind of plot.

The first problem concerns the definition of similarity. The similarity may be calculated in the decision or in the objective space. For calculation in the decision space, the simplest way is to calculate the inverse Hamming distance (number of identical decision variables). With this measure, the problem arises that a certain amount of randomness is introduced. As solutions are grouped because of some modules, the rest of the decision variables (which are not in these modules) may or may not be similar but still contribute to the Hamming distance.

To even out the influence of erroneous ones, the similarities are calculated for each pair of solutions in the group. Then, the average over these similarities of pairs is taken.

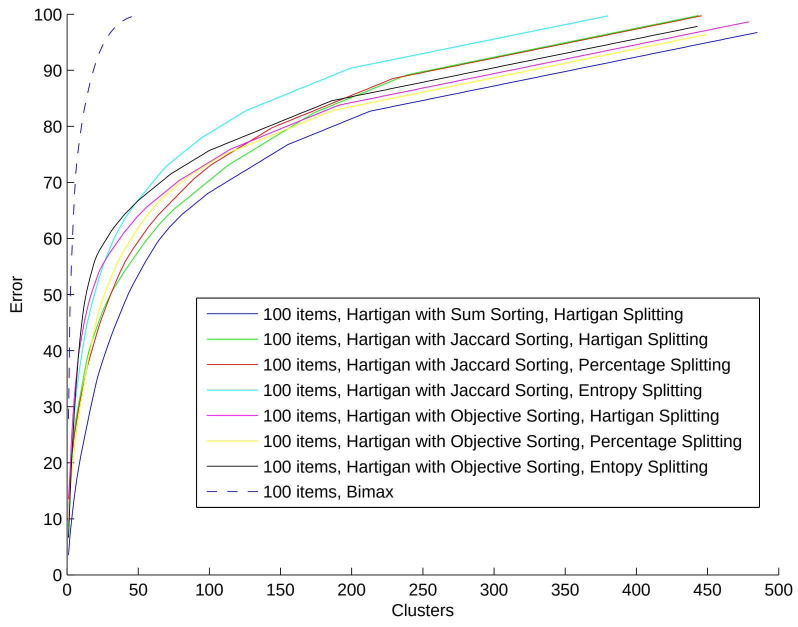


Figure 2.4: Example Structure/Error Plot for Pareto-optimal Front with 100 Items

This measure leads us to the next problem: The Hartigan and the Bimax algorithms are difficult to compare, as the Bimax algorithm has the advantage of finding larger biclusters, because the Bimax biclusters are allowed to overlap.

An example plot of such a group/similarity plot may be seen in Figure 2.5. Here, the similarity is calculated as the average pairwise inverse Hamming distance of the group.

### 2.6.3 Comparison

In the structure/error plot, the Bimax algorithm obviously performs better than the Hartigan algorithm, because the Bimax algorithm produces fewer and larger biclusters. Consequently, the ones in the matrix are covered with fewer modules than when using the Hartigan algorithm.

In the group/similarity plot, the Bimax algorithm, at least with the increasing grouping method, also performs slightly better than the Hartigan algorithm, i.e. it produces fewer groups while maintaining similarities comparable to those of the Hartigan algorithm. Again, this may be attributed to the fact that the Bimax algorithm produces fewer biclusters because they are allowed to overlap.

The problem of the structure/error plot obviously is that the similarities of groups are not considered. The group/similarity plots, on the other hand, are calculated based on all modules, thereby eliminating any possibility of allowing error in the matrices.

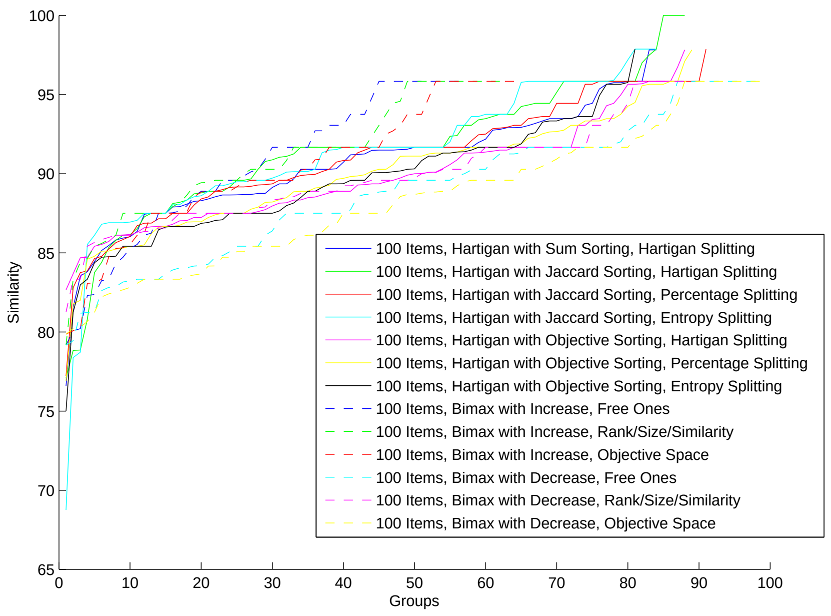


Figure 2.5: Example Group/Similarity Plot for Pareto-optimal Front with 100 Items

Therefore, a new approach will be devised in the next section, where the two plots are connected.

### 2.6.4 Best Level Cut Based on Both Plots

To combine both plots into one, it was decided to forget about the different group similarities and use only the average similarity of all groups instead.

With that decision made, the error can be plotted against the average similarity of the groups that are generated by the modules that correspond to the respective error. In other words, to generate the new plot, the structure/error plot is traversed from right to left, the error increasing as more and more modules are discarded. In each step, the groups are calculated using the remaining modules and their average similarity is plotted against the error (see Figure 2.6 for an example).

To find a best level cut on this plot, a method has to be found that is not sensitive to minor irregularities in the plot, which occur especially at the beginning of the plot. This is the reason why methods that consider the maximum gradient do not deliver consistent results.

Therefore, another method is devised: The plot is normalized such that its lower left point starts at (0,0) and its upper right point ends at (1,1). Then, a straight line with a pre-defined gradient which passes through the leftmost point of the plot is considered. The best level cut is the point which is furthest

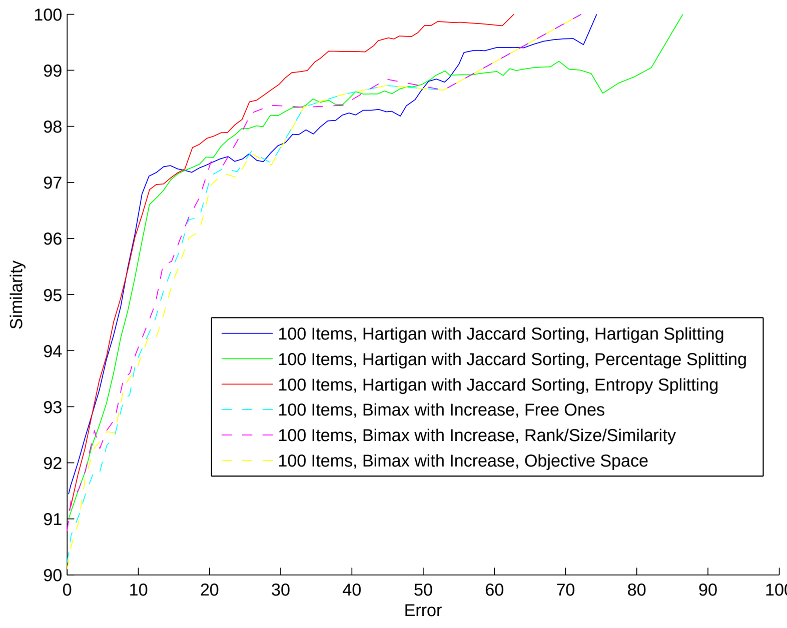


Figure 2.6: Example Error/Similarity Plot for Pareto-optimal Front with 100 Items

away (Euclidean distance) from this line (see Figure 2.7 for an example). By default, the gradient is 45 °. When aiming for a lower error, the gradient should be higher. For a higher similarity the gradient should be lower.

## 2.7 GUI

A Matlab based general user interface (GUI) was developed to provide an easy way of testing the different biclustering methods on a matrix. It is possible to select a matrix and a biclustering and grouping method. The resulting groups are automatically displayed in the objective space.

For further analysis, an error may be set and the resulting biclusters can be displayed in the decision space. To investigate about single groups, their solutions can be displayed in decision space. Additionally the decision variables with are selected in all or none of the solutions of this group are displayed. It is also possible to find the best level cut.

An introduction on how to use the GUI is given in Appendix A.

## 2.8 Summary

When optimizing complex systems, one of the problems that emerge is that after the search, when the Pareto-set approximation is given, it is difficult to

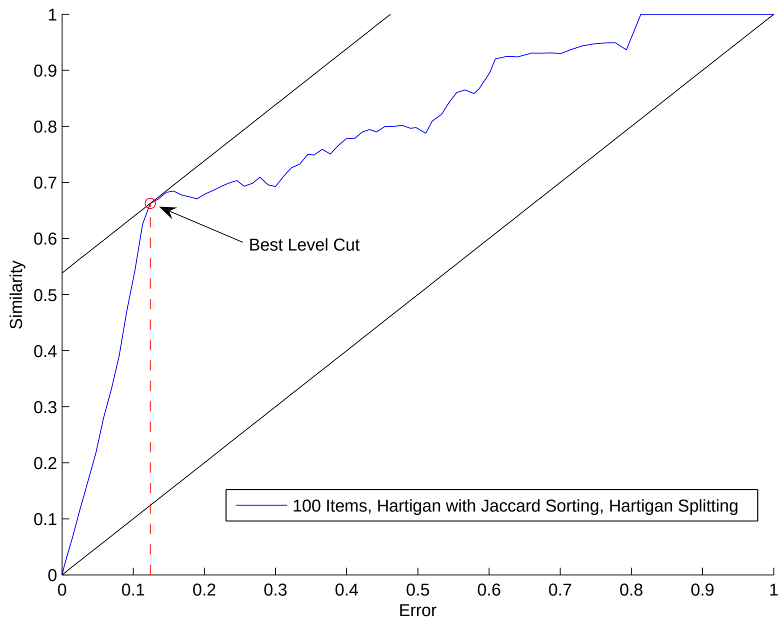


Figure 2.7: Example for Finding the Best Level Cut for a Pareto-optimal Front with 100 Items

decide on a final set of solutions. In the offline scenario, methods have been developed which find modules to highlight similarities of decision variables and which group solutions with similar modules.

To find modules automatically, two biclustering algorithms, namely the Hartigan and the Bimax algorithm, have been used. For both algorithms, several variations exist. In case of the Hartigan algorithm, these variations focus on the identification of modules. In case of the Bimax algorithm, on the other hand, different methods have been developed to group solutions when given a set of modules.

The grouping of solutions results in a tree structure of groups. For more complex systems, this tree becomes very large. As the user should not be presented with the whole tree, a method which cuts the tree on the best level has been developed.

Finally, a GUI has been implemented which makes it possible to test the algorithms on different problems.

# Chapter 3

# Offline Scenario Results

In the last chapter, various methods for finding modules in binary decision space matrices and for grouping solutions with similar modules have been developed. In this chapter, these methods will be compared in multiple test cases and the best method will be selected. More tests will then be run on this selected method to gain a general idea on how well the representation reduction can be done by applying the selected method. Finally, the method will be applied to a network processor design to test its usability in real-world applications.

## 3.1 Parameters

As mentioned before, the considered test problem is the two-dimensional knapsack problem. The implementation of the knapsack problem was taken from the PISA framework [17]. Table 3.1 describes the parameters that were used for the knapsack implementation of PISA.

| Parameter | Value |
|---|---|
| Capacity | 0.4 * weight sum of all items |
| Recombination | one point crossover |
| Recombination probability | 0.5 |
| Mutation | one bit mutation |
| Mutation probability | 1 |

Table 3.1: Parameters for Offline Problem

The different methods are compared according to several measures, as described in the next sections. First, the methods are applied to test problems to verify the expected results. Then, the influence of the number of decision variables on the grouping results is investigated. With these test results, one method is chosen which will be used for all subsequent tests.

This selected method is then applied to random matrices, which yields reference values for the subsequent tests. Next, the same problem will be tested with different seeds to gain information about the influence of the problem structure on the test results. Furthermore, the differences between actual Pareto-optimal fronts and Pareto-optimal front approximations are examined.

Finally, the method is applied to a realistic network processor design problem [16].

## 3.2 Method Selection

There are numerous ways to calculate both modules and groups, but only certain combinations make sense. The following two sections will explain the criteria based on which a subset of these combinations is selected.

### 3.2.1 Selection of Biclustering Methods

First, the Hartigan method is used to calculate modules and groups. There are several different ways to find the modules (see Sections 2.4.1 and 2.4.2). In this thesis, it was decided to test only a selection of all possible combinations. As illustrated in Section 2.4.1, sorting according to row and column sums yields poor results for binary matrices. Nevertheless, the split measure proposed by Hartigan theoretically produces the best splits for such a sorting. Therefore, sorting according to row and column sums will be tested, but only with the split measure proposed by Hartigan.

Three sorting measures remain: Sorting according to Hamming or Jaccard distance, and sorting according to objective space values. As sorting according to Hamming distance is quite similar to sorting according to Jaccard distance, it makes sense to only use one of these two measures. The Hamming distance calculates the percentage of differing bit-pairs in relation to all bit-pairs, whereas the Jaccard distance calculates the percentage of differing bit-pairs in relation to those pairs whose bits contain at least a one. As in this thesis, modules are defined as biclusters of ones, only the Jaccard distance will be used for this chapter. It will be tested with all three split measures (see Section 2.4.2).

The sorting according to objective space values is quite different from the other two sorting measures, therefore it will also be tested with all three split measures. As the objective space values only allow for a sorting of the y-axis, the x-axis has to be sorted indepently. For reasons of conformity, the Jaccard distance measure will be used.

### 3.2.2 Selection of Grouping Methods

As the Hartigan method produces a tree on the y-axis, only one possible grouping method exists. Bimax on the other hand only uses one biclustering method, but may apply several grouping methods as explained in Section 2.5.3.

Again, it makes sense to only use a selection of all possible combinations. Both the increasing and the decreasing method will be used, therefore the selection has to be made on the definition of the largeness of modules.

It was decided to select two basic methods, one of which defines the largeness of modules in decision space, the other one in objective space. In addition to these two basic methods, the more sophisticated method based on rank, size and similarity in objective space will be considered.

For the basic methods in decision space, it was decided to use the number of free ones, i.e. ones which are not yet covered by other modules to measure the largeness of a module. Corresponding to this, the largeness in objective space will be defined as the objective space deviation that results from discarding those free ones, measured by the sum of differences of the Euclidean distances of all solutions.

### 3.2.3 Overview of Selected Methods

Table 3.2 gives an overview over the selected methods.

| **Method** | **Algorithm** | **Variations** | |
|---|---|---|---|
| Biclustering Method | Hartigan | **Sorting** | **Split Measure** |
| | | Row & column sums | Hartigan |
| | | Jaccard distance | Hartigan<br>Percentage<br>Entropy |
| | | Objective space | Hartigan<br>Percentage<br>Entropy |
| Grouping Method | Bimax | **Direction** | **Bicluster Size Definition** |
| | | Increase | Decision space<br>Objective space<br>Rank, size & similarity |
| | | Decrease | Decision space<br>Objective space<br>Rank, size & similarity |

Table 3.2: Methods Overview

### 3.2.4 Selection of Best Method

For most of the tests, only the best method will be used. To find the best method, they are first verified with constructed test matrices (see Section 3.3.2), where the general behavior of the methods may be studied.

Then the methods are applied to several Pareto-optimal fronts that differ in the number of items (see Section 3.3.3).

The results of both tests may be found in Sections 3.3.2 and 3.3.3. This section summarizes the results of the following sections.

Using the test matrices it was found that sorting according to row and column sums yields poor results and therefore will not be considered further. The results of the rest of the algorithms was acceptable, where none of the algorithms performed considerably better than the others.

Therefore, to select a final algorithm from the remaining ones, the real matrices with different items were compared. On the whole it can be said that the original

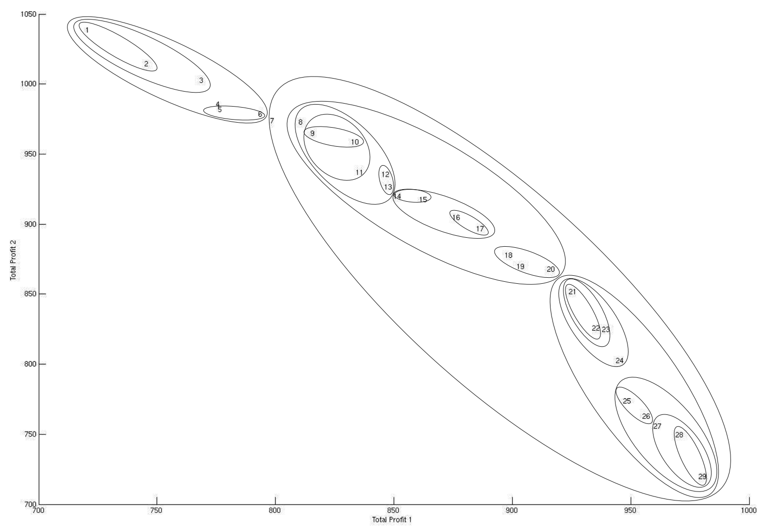


Figure 3.1: Grouping in Objective Space

Bimax algorithm is very limited in terms of matrix size. As the Bimax algorithm finds all possible biclusters, the number of biclusters increases exponentially with matrix size. Increasing the minimum matrix size does not improve matters, as only biclusters with enough free ones, i.e. ones not yet covered by other biclusters, are selected. These selected biclusters are usually not the largest biclusters, as large biclusters also have a higher overlap. For this reason, the error increases considerably if the minimum bicluster size increases. Due to this matrix size restriction, Bimax will not be considered further.

After discarding the Bimax algorithm, only the Hartigan algorithm remains. As sorting according to row and column sums has been discarded, the only sorting measures that remain are the Jaccard measure and sorting according to objective space values. As sorting according to objective space values per definition strongly depends on the distribution of the objective space values, only the Jaccard measure will be considered. It has to be noted that if groups that are close in objective space are desired, sorting according to objective space values is the best choice. Figure 3.1 shows such a grouping with sorting according to objective space values. For other sorting measures, the groups usually do not only encompass neighboring solutions.

In a last stage a split measure has to be chosen. The entropy split measure is discarded, as it does not yield better results but still takes considerably longer to calculate than the calculation of modules with the Hartigan or the Percentage split measure (see Section 3.3.3).

The decision between the Hartigan and the Percentage split measure is more difficult to make, as the results of these two split measures are quite similar. In the end, the decision was made in favor of the Hartigan split measure, as it produces better results for the largest matrix.

## 3.3 Tests

### 3.3.1 Testing Overview

This section gives an overview over the characteristics that are tested. For each Pareto-optimal front, a selection of characteristics is calculated. First, the biclustering is done using the methods listed in Table 3.2. For the resulting modules, the following data is calculated:

- Calculation time for module calculation
- Number of modules
- Average height of modules
- Average width of modules
- Average size of modules
- Error of modules in decision space
- Error of modules in objective space

The error of a module in decision space is defined as the number of ones of this module that are not yet covered by any other module. The error in objective space is the deviation resulting from discarding these free ones, which is the sum of Euclidean distance differences of all solutions. As this error depends on the order in which the modules are selected, the modules will be ordered from largest to smallest. Two variations exist, as they will either be ordered according to the error in decision space or according to error in objective space.

The error calculated for the modules will next be used to group the solutions. Modules will iteratively be discarded from smallest to largest module. For each of the resulting errors, the following group characteristics will be calculated:

- Calculation time for grouping
- Number of groups
- Average height of groups
- Average similarity in decision space of groups
- Average similarity in objective space of groups

Several ways to calculate similarity in both decision and objective space exist. It was decided to use two comparable measures. In decision space, the similarity of a group is defined as the average pairwise Hamming distance, whereas in the objective space it is the average pairwise Euclidean distance. Both measures are normalized to the interval $[0, 100]$.

#### Plot Explanations

This section contains explanations regarding the matrix plots, where matrices, their modules and the corresponding groups are shown. The main part of the plot is the matrix itself. To enhance presentability, the ones are represented by dark blue squares, whereas the zeros are represented by light blue squares. In these matrices, the modules are plotted with black rectangles in case of the Hartigan algorithm and with colored rectangles in case of the Bimax algorithm. On the right side of these plots, the groups are drawn. Each row group is represented by one vertical line. In case of groups over non-subsequent rows, the part of the skipped rows is indicated by a dotted line.

### 3.3.2 Predefined Test-Matrices

First, the methods are compared by applying them to constructed test matrices. These test matrices have clearly defined biclusters which the methods should be able to identify.

Two types of predefined test-matrices will be used. The first type contains its biclusters only in the diagonal, which ensures that biclusters do not overlap in either axis. The second type has randomly distributed biclusters, with the constraint that biclusters cannot overlap.

Each matrix has two sets of random objective space values, where the first one is sorted in ascending order according to the original matrix row order, and the second one is sorted randomly. Note that these objective space values are only used by those Hartigan algorithms that sort the matrix according to objective space values.

First, only the biclustering and grouping techniques are compared, in which case the matrices will be used without permutation. For the comparison of the method as a whole, the matrices will be permuted prior to biclustering, making the sorting technique an important part of the method.

#### Biclusters in Diagonal

**Not Permuted** Matrices with the biclusters in their diagonal are the simplest case. To compare the split measures, the matrix is not permuted prior to bicluster calculation. Therefore, the sorting method has no influence on the results.

In one test run, it has been found that both the Hartigan and the Bimax algorithm find all biclusters correctly, and each bicluster makes up one group. The group similarities are always 100%, because solutions inside single groups look exactly the same. Figure 3.2 shows the test matrix with the found biclusters and groups. This plot looks the same for all biclustering and grouping algorithms.

**Permuted** In a next step, the sorting methods are considered as well. In the test run it has been found that the measure which sorts according to row and column sums (sum sorting measure) is not able to sort a decision space matrix correctly, if several biclusters have identical row or column numbers.

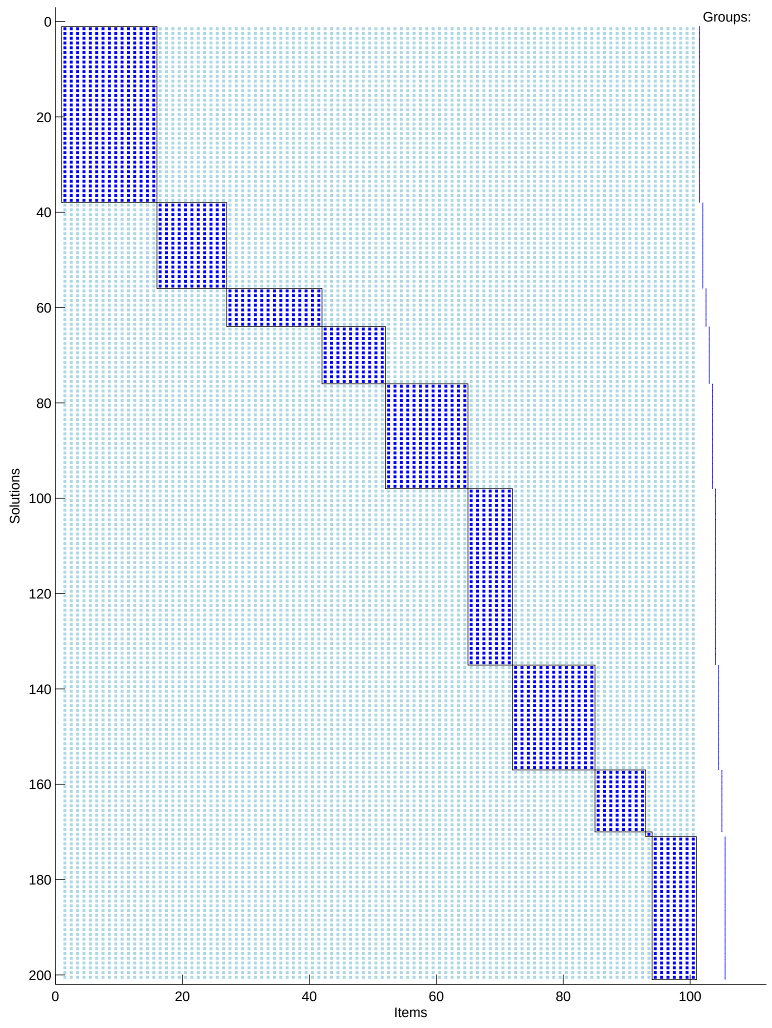


Figure 3.2: Bicluster and Group Results for all Algorithms

The Jaccard sorting measure, on the other hand, sorts the matrix correctly. The result of the measure that sorts according to objective space values obviously depends on the chosen objective values. If these objective space values are in the correct order, the sorting is done correctly. If they are permuted randomly, only the columns are sorted correctly, as the sorting according to objective space values only applies to row sorting, whereas column sorting is done with the Jaccard measure.

The results for the Bimax algorithm are the same as in the non-permuted case, as sorting has no influence on Bimax.

Figure 3.3 shows the biclusters and groups of the test matrix found by the sum sorting measure, Figure 3.4 shows the results for sorting according to objective space values, in the case that the objective space values are permuted randomly.

Bimax and the Hartigan algorithm with Jaccard sorting measure and sorting according to objective space values, where those objective space values are not permuted, produces the same results as shown in Figure 3.2

### Randomly Distributed Biclusters

In this section, the biclusters are randomly distributed, with the constraint that they cannot overlap in the matrix itself. But as the biclusters do overlap in both axes, the best groups are not evident anymore.

**Not Permuted** In this first subsection, only the split measures are compared. The matrix is not sorted prior to biclustering. As the results of all split measures look similar, although not identical, only the results for the Hartigan split measure are shown (see Figure 3.5).

Bimax finds all biclusters of this test matrix. But depending on the applied grouping algorithm, different groups are found. The grouping with bicluster size definition in objective space cannot be calculated for the test matrices, as the profits of the items are not given. The results of the remaining grouping measures again look similar and only the increasing grouping method with size defined in decision space is shown (see Figure 3.6).

In comparison to the Hartigan algorithm results, the Bimax groups do encompass solutions that do not necessarily lie next to each other. As a result, the calculated groups are on average larger than those found by the Hartigan algorithm.

**Permuted** In the case where the matrix is permuted prior to biclustering, the different sorting methods are used for the Hartigan algorithm, whereas the results of the Bimax algorithm do not change, as they are independent of the sorting of the matrix.

The results for the sum sorting with Hartigan split measure is shown in Figure 3.7, those for the Jaccard sorting with Hartigan split measure are shown in Figure 3.8.

The results for the sorting according to objective space values for non-permuted objective space values with Hartigan split measure are shown in Figure 3.9,

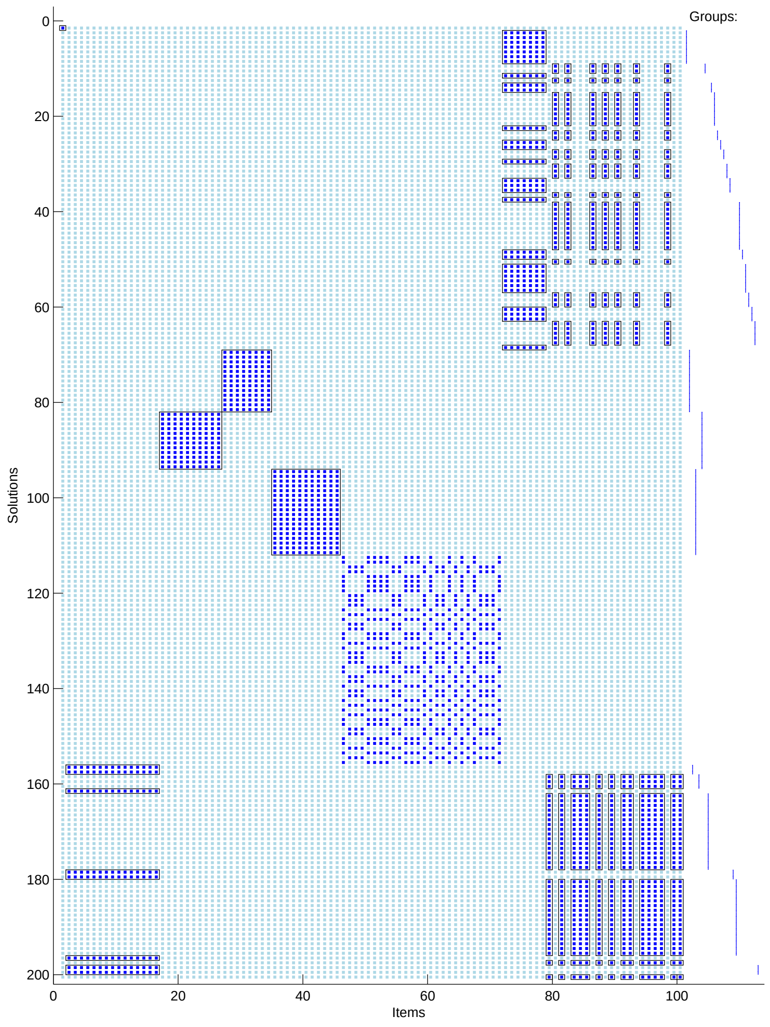


Figure 3.3: Bicluster and Group Results of the Hartigan Algorithm with Sum Sorting and Hartigan Splitting

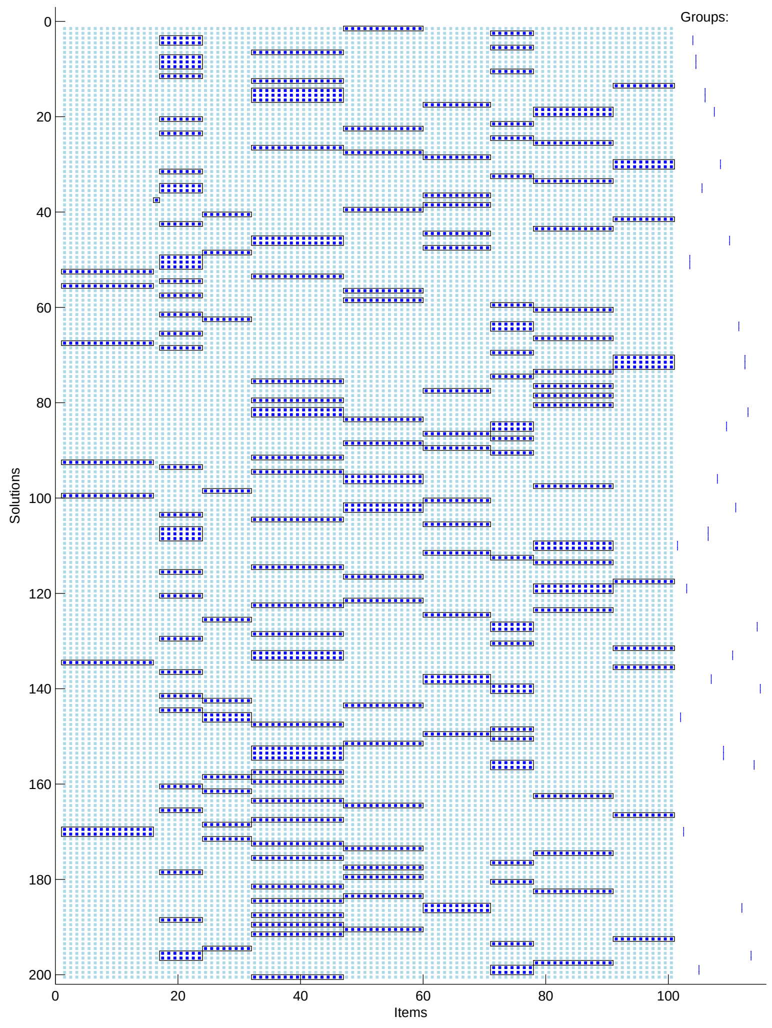


Figure 3.4: Bicluster and Group Results of the Hartigan Algorithm with Sorting According to Objective Space Values, where those Values are Permuted Randomly

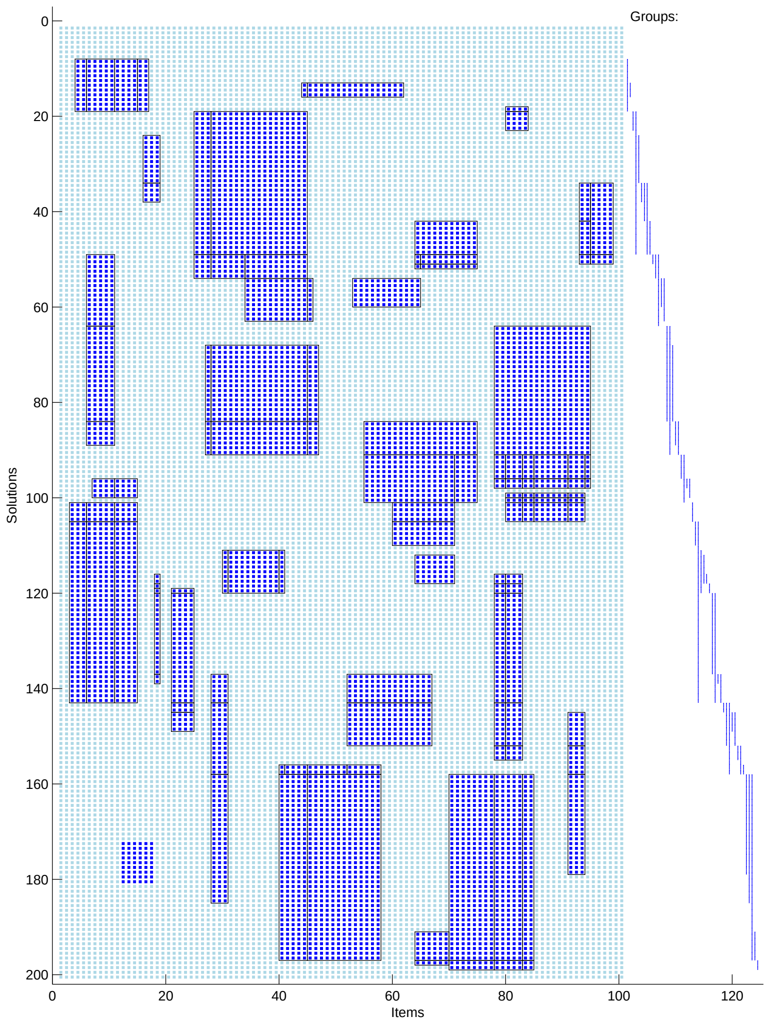


Figure 3.5: Bicluster and Group Results for the Hartigan Algorithm with Hartigan Splitting

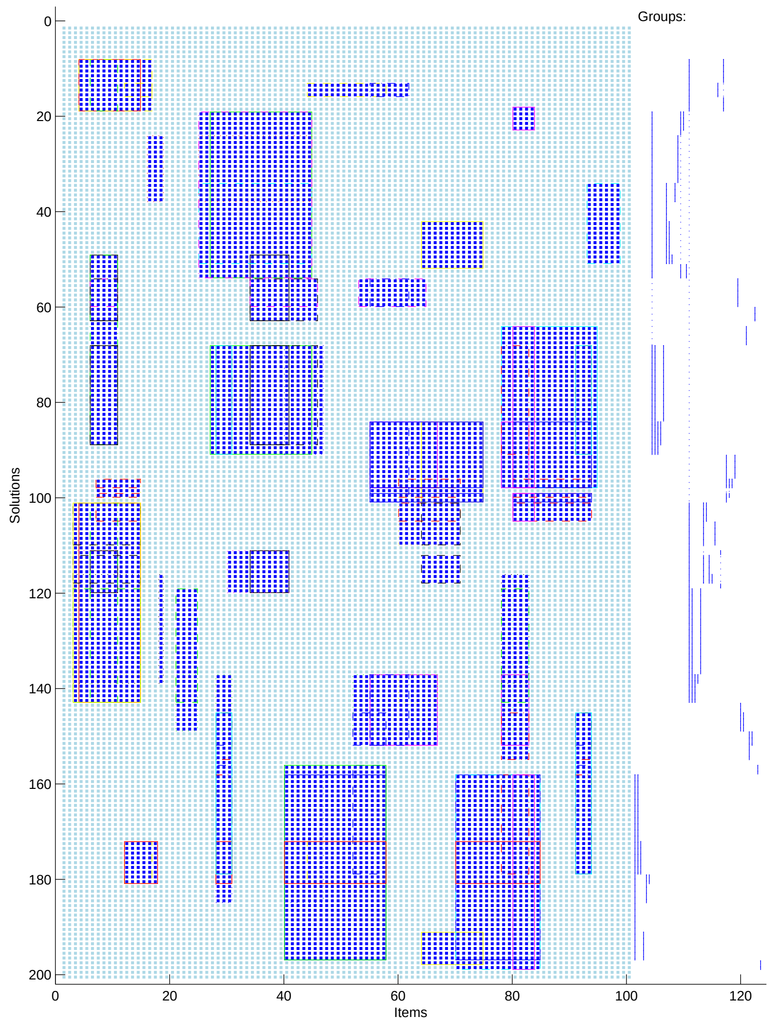


Figure 3.6: Group Results for Bimax Algorithm with Increase Method and Size in Decision Space

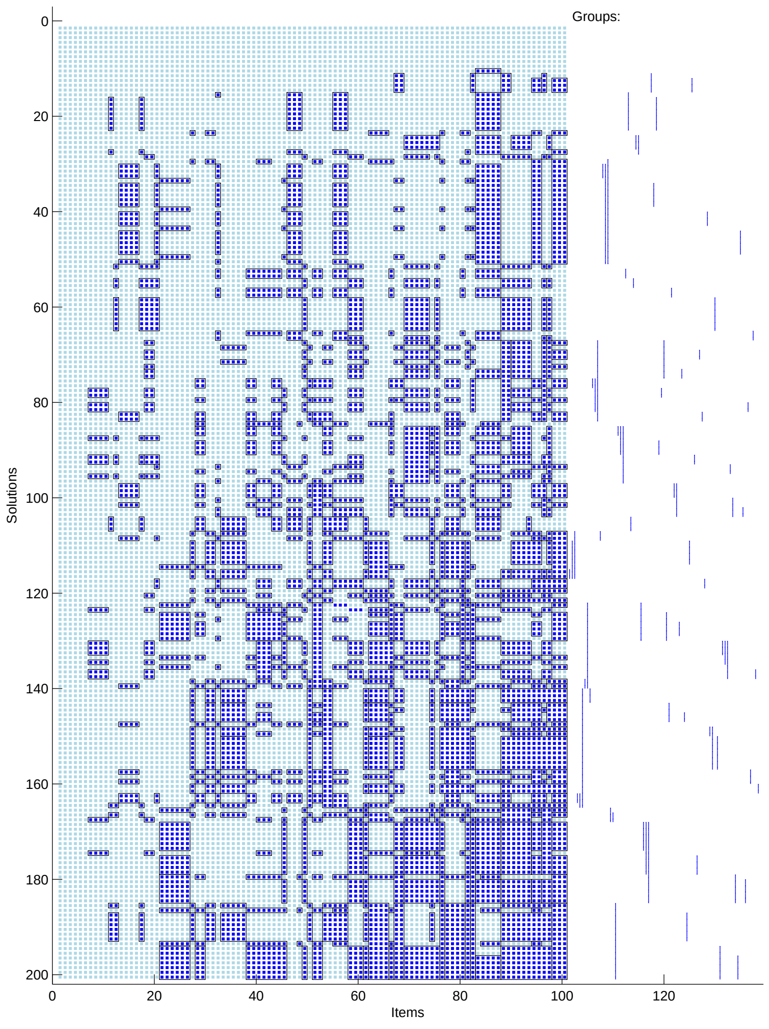


Figure 3.7: Bicluster and Group Results for the Hartigan Algorithm with Sum Sorting and Hartigan Split Measure

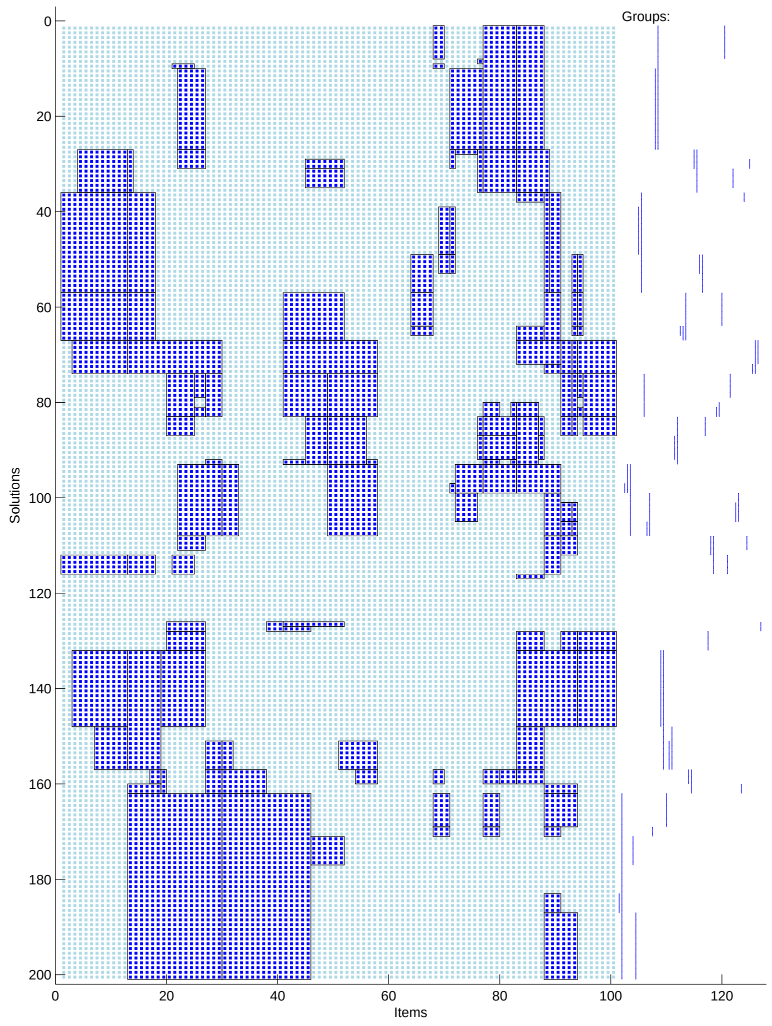


Figure 3.8: Bicluster and Group Results for the Hartigan Algorithm with Jaccard Sorting and Hartigan Split Measure

whereas the results for sorting according to randomly permuted values with Hartigan split measure are shown in Figure 3.10.

It can be seen that while sorting according to row and column sums, as well as sorting according to randomly permuted objective space values yields unacceptable results, the results for Jaccard sorting and sorting according to non-permuted objective space values are comparable and yield good results.

### 3.3.3 Number of Items

In this section, the biclustering and grouping methods are applied to real Pareto-optimal fronts. Different optimization problems that vary in the number of items are tested. 4 problems are considered, with 50, 100, 150 and 200 items and with seeds 1, 2, 3 and 4 respectively. The use of Pareto-optimal fronts ensures independence of of approximation algorithms.

For matrices with more than 200 items, the computation time of both Hartigan and Bimax algorithm becomes too high.

The limit of the Hartigan algorithm is given by the number of splits that are required to partition the matrix into biclusters that only contain ones or zeros. The larger the matrix, the more splits are required. To alleviate the problem of growing split numbers, a minimum bicluster size $s_{min}$ may be introduced, such that a bicluster with less than $s_{min}$ elements will not be split again even if it still contains both ones and zeros. The ones contained in these biclusters are considered as error.

The limit of the Bimax algorithm, on the other hand, is given by the number of biclusters that are found. To move on to the grouping, a subset of these biclusters has to be selected, which ideally covers all ones in the matrix. To assure a good coverage, the selection of the $i+1$-th bicluster depends on the $i$ previously selected biclusters, thereby giving this selection process a complexity of $\mathcal{O}(n^2)$, where $n$ is the total number of biclusters. Similarly to the Hartigan algorithm, a minimum bicluster size may be chosen for the Bimax algorithm to decrease the number of biclusters that have to be considered. Ones that are only covered by smaller biclusters are declared as error.

Although the setting of a minimum bicluster size is perfectly suitable for real life applications, it cannot be used for testing and comparison of methods. There is no easy rule as to how to set such a minimum bicluster size in a fair way.

#### Results for Different Numbers of Items

In each test case, four different similarity plots are calculated. In each similarity plot, the similarity, which may either be calculated in decision or in objective space, is plotted against the respective error. The errors are calculated by subsequently declaring the smallest remaining bicluster as error. The more biclusters are discarded, the higher the error and the higher the similarity is. The size of a bicluster may either be defined in the decision or in the objective space.

The four plots therefore are grouped into pairs of two, where the upper pair has the biclusters discarded in order of their size in decision space, and the lower

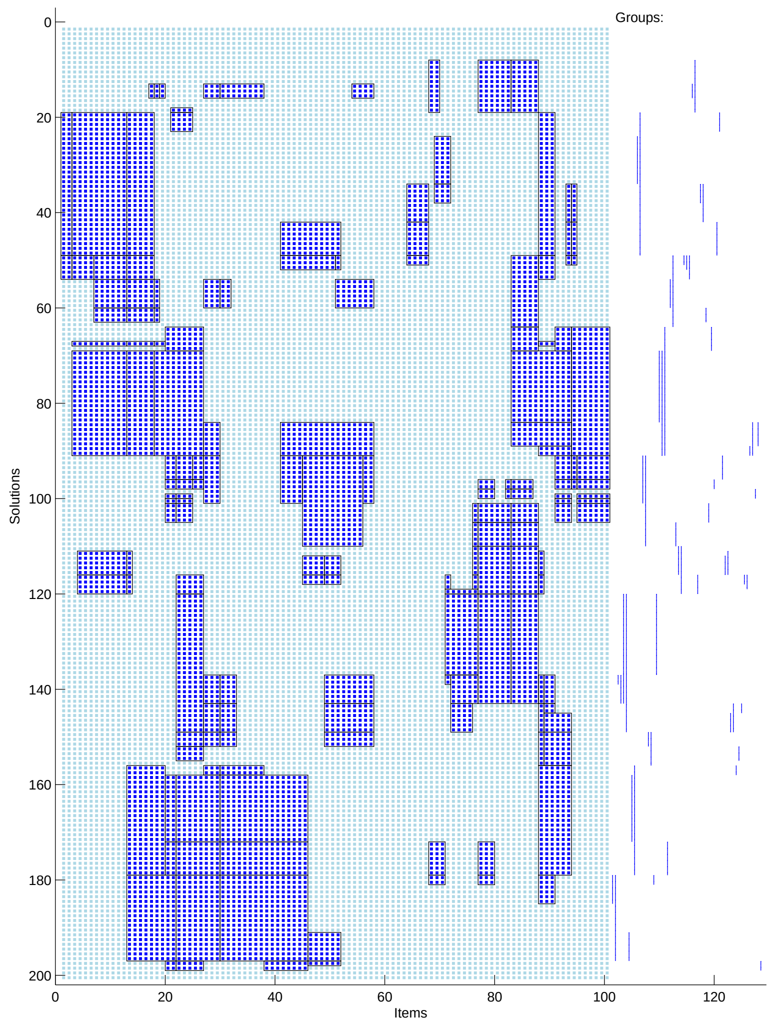


Figure 3.9: Bicluster and Group Results for the Hartigan Algorithm with Sorting According to non-Permuted Objective Space Values, with Hartigan Split Measure

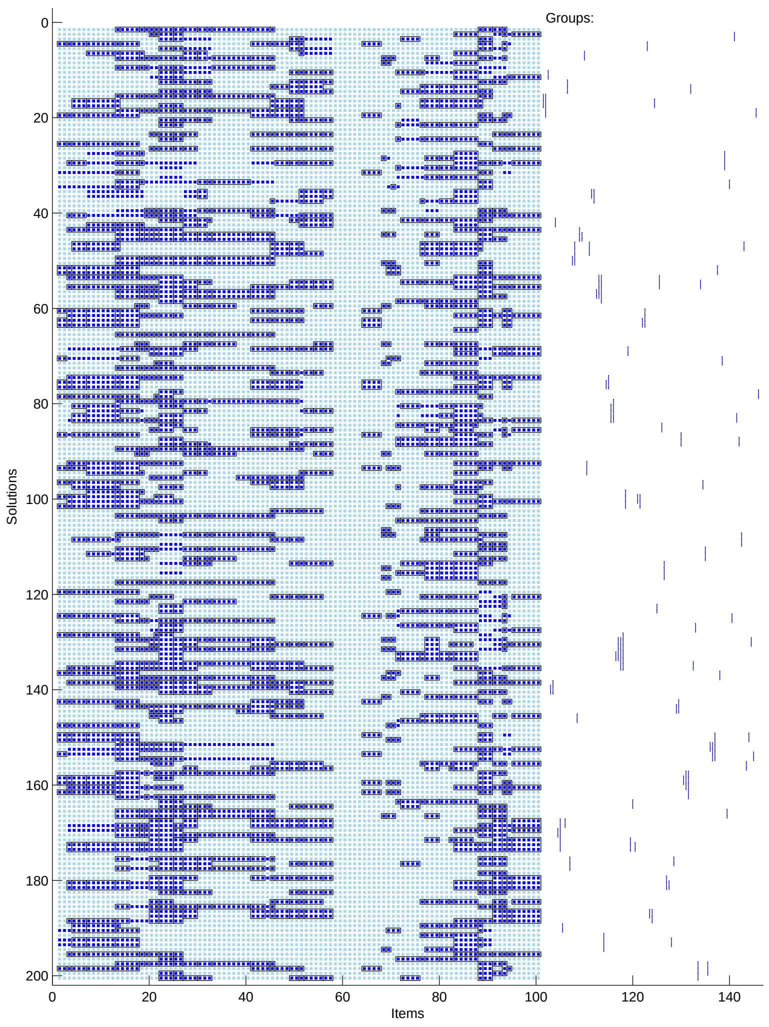


Figure 3.10: Bicluster and Group Results for the Hartigan Algorithm with Sorting According to Randomly Permuted Objective Space Values, with Hartigan Split Measure

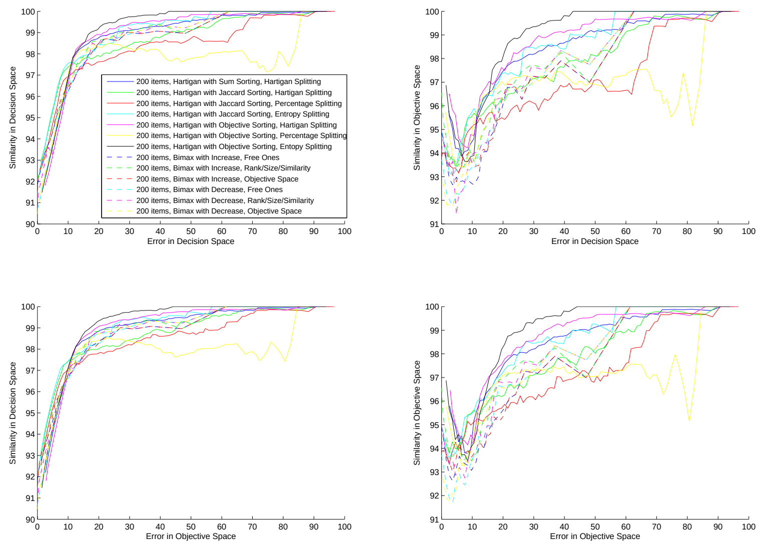


Figure 3.11: Different Similarity Plots for Problem with 200 Items

pair calculates the bicluster size in objective space. Each pair consists of two plots, where the similarity is calculated in decision space in the first one and in objective space in the second one.

These four plots in case of the matrix with 200 items is shown in Figure 3.11. For a low error, it can be seen that the Jaccard sorting produces groups with a higher similarity in decision space, whereas the sorting according to objective space values yields higher group similarities in objective space. As the error grows, this effect vanishes and the order of the methods is the same in each plot. For the remaining matrices, only the plots are used where the error is calculated by discarding biclusters based on their size in decision space, and where the similarity is also calculated in decision space.

As the Bimax algorithm is the limiting factor for the matrix size (although the Hartigan algorithm is not far behind), it will not be considered further. The Hartigan algorithm with sorting according to row and column sums will also be discarded, due to the poor results obtained from test matrices (see Section 3.3.2). Sorting according to objective space values will also be discarded, as described in Section 3.2.4.

The similarity plots for the remaining three algorithms is shown in Figures 3.12 to 3.15. As the similarity in decision space is calculated as the average pairwise Hamming distance, it is closely related to the average height of groups. Larger groups usually have a higher average pairwise Hamming distance. To take this dependence into account, the average height of groups is plotted next to the similarity plot.

In these plots it can be seen that the groups found by the Entropy split measure are generally smaller than those of the other two split measures. The larger the

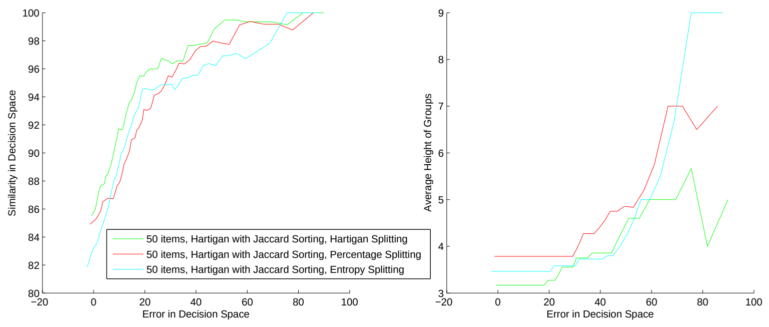


Figure 3.12: Similarity Plot for 50 Items

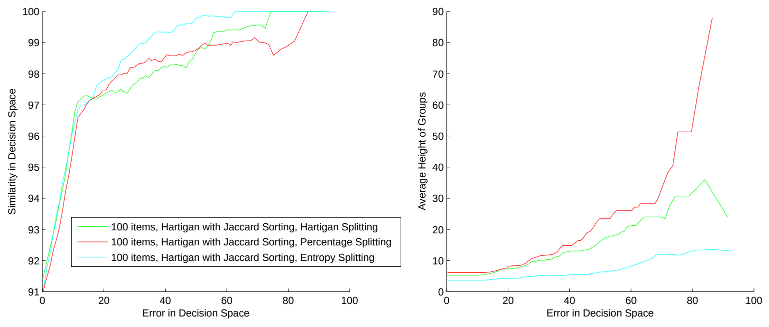


Figure 3.13: Similarity Plot for 100 Items

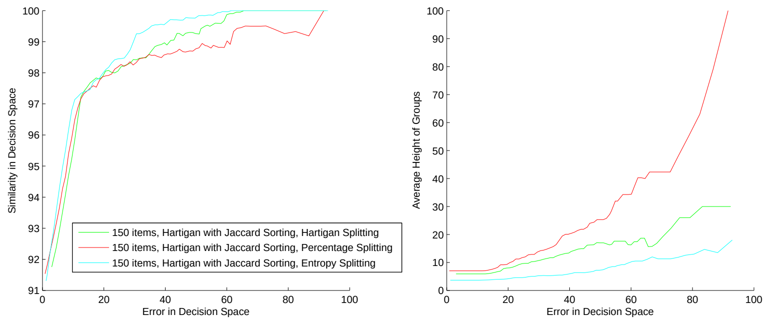


Figure 3.14: Similarity Plot for 150 Items

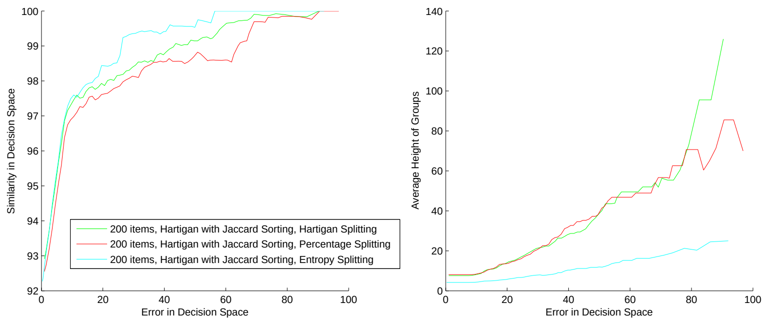


Figure 3.15: Similarity Plot for 200 Items

difference, the better are the results in the corresponding similarity plot, which is a direct consequence of the small groups. In Figure 3.12, where the group size of the Entropy split measure in the error interval $[20, 60]$ is approximately the same as for the Hartigan split measure, it performs considerably worse in the similarity plot. When taking into account that the Entropy splits take much longer to calculate than the other two splits[1], the results are not enough superior to the results of the other two split measures to consider the Entropy split measure any further.

To decide between the Hartigan and the Percentage split measure, a closer look at the similarity plots has to be taken. For the similarity plots of the matrices with 50 and 150 items, the Hartigan split measure performs better, but at the same time finds smaller groups. As this yields no information about the true superiority of the measures, the other two plots have to be considered. In the similarity plot of the matrix with 100 items, the Percentage split measure shortly performs better even though it finds slightly larger groups. In the similarity plot of the matrix with 200 items, on the other hand, the Hartigan split measure constantly performs slightly better even though the groups are of the same size as those found by the Percentage measure.

In the end, the Hartigan split measure is chosen, keeping in mind that its results are comparable to those of the Percentage split measure.

### 3.3.4 Random Matrices

From now on, only the best algorithm, which in Section 3.2.4 was found to be the Hartigan algorithm with Jaccard sorting and Hartigan split measure, will be tested.

In this section, the selected algorithm is run on random matrices. The goal is to ascertain the degree of structure of random matrices, which will serve as a reference for the performance of these algorithms on Pareto-optimal fronts or Pareto-set approximations.

[1]The entropy calculation for a row split is linearly dependant on the number of columns of the bicluster and therefore the split calculation is directly proportional to the number of rows times the number of columns of the bicluster.

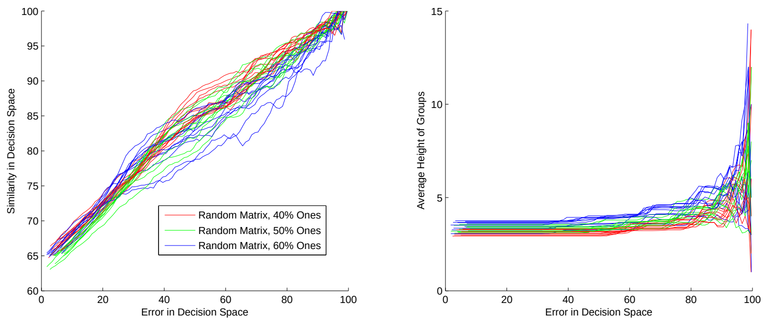


Figure 3.16: Similarity Plots for Random Matrices

The random matrices will be divided in three sets. These sets differ in the percentage of ones that their matrices contain. The matrices of these three sets consist of 40%, 50% and 60% ones, respectively. Each set contains 11 matrices. The size of these matrices was chosen to be the average size of the matrices of Section 3.3.5, which is 150x55.

The similarity plots of all random matrices are shown in Figure 3.16. For an error of zero, the average inverse Hamming distance of groups found by the Hartigan algorithm is 65% of all decision variables. The similarity then increases linearly with increasing error until both the error and the similarity reaches 100%[2]

It can also be seen that the group sizes are slightly larger for those matrices with 60% ones, as more ones produce larger biclusters and therefore larger groups. As a result, the similarities of matrices with more ones are slightly lower than those of matrices with less ones and smaller groups.

In conclusion, the average similarities of groups calculated on random matrices vary between 65% and 100%, whereas the similarities of solution groups of Pareto-optimal fronts vary between 90% and 100%. This leads to the conclusion that Pareto-optimal fronts indeed are highly structured.

### 3.3.5 Different Seeds

In this section, the influence of the problem itself, defined by its seed, is investigated. To this end, 11 Pareto-optimal fronts with 100 items and seeds 1 to 11 are tested. The similarity plots for all of these matrices, calculated again by the Hartigan algorithm with Jaccard sorting and Hartigan split measure, are shown in Figure 3.17.

Both the resulting similarity plots and the average group sizes vary strongly for different seeds. On one hand, this is expected, as different seeds yield completely different matrices. On the other hand, it shows that the representativeness of single test runs, for example for method comparison, is rather low.

[2]If all ones in the matrix are declared as error, the similarity of all solutions obviously is 100%

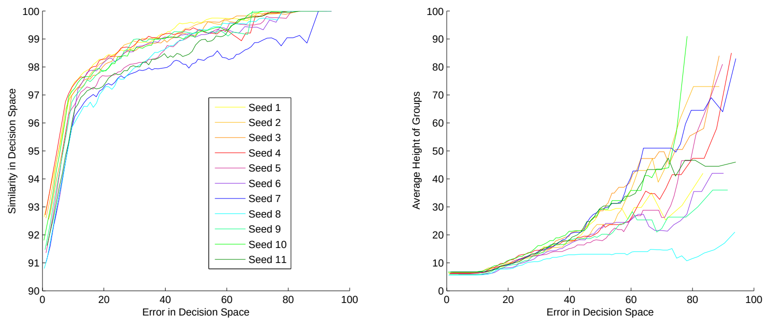


Figure 3.17: Similarity Plots for Different Seeds

### 3.3.6 Pareto-optimal Front vs. Approximation

In this section, the differences between groups calculated on the Pareto-optimal front and groups calculated on an approximation of the Pareto-optimal front are examined. For the approximation, the IBEA algorithm is used. Therefore, the IBEA algorithm is run on the optimization problems used in Section 3.3.5. The IBEA algorithm uses population sizes 100 and 300, each time with 500 generations. Again, a knapsack problem with 100 items is considered.

The differences between the Pareto-optimal front and the approximation with a population size of 100 can be seen in Figure 3.18. In the right plot of this Figure it can be seen that the average group sizes found in the approximation are smaller than the average group sizes found in the Pareto-optimal front. Even though this suggests that the similarities should be higher for the groups of the approximation matrix[3], they are not, as can be seen in the left plot of the Figure.

Interestingly enough, the similarities for the approximation with population size 300 are slightly better than those of the approximation with population size 100, even though the average group sizes are larger for the approximation with population size 300 than for the approximation with population size 100. This is shown in Figure 3.19.

The similarities of groups of the approximation with population size 300 are also slightly better than those of the Pareto-optimal front (see Figure 3.20), but this may be expected as the group sizes of the approximation are smaller than those of the Pareto-optimal front.

### 3.3.7 Application on Larger Matrices

The problem with larger matrices is that the Matlab calculations are quite slow. The sorting procedure alone is quadratically complex in the number of rows and columns.

Figure 3.21 shows the calculation times of the Hartigan algorithm for different numbers of items and different minimum bicluster sizes.

[3]The similarity is calculated as the average pairwise Hamming distance in decision space

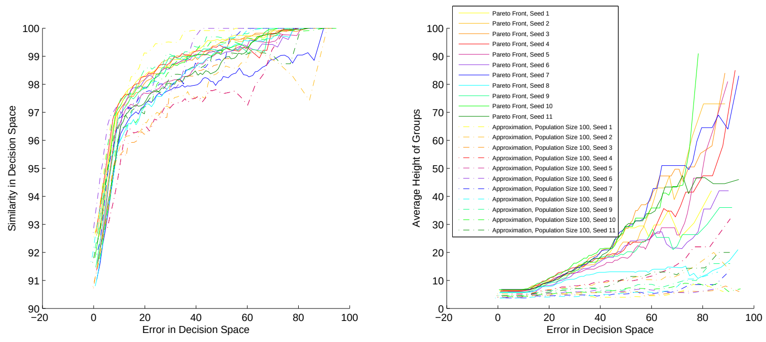


Figure 3.18: Similarity Plots for Pareto Fronts and Approximations with Population Size 100

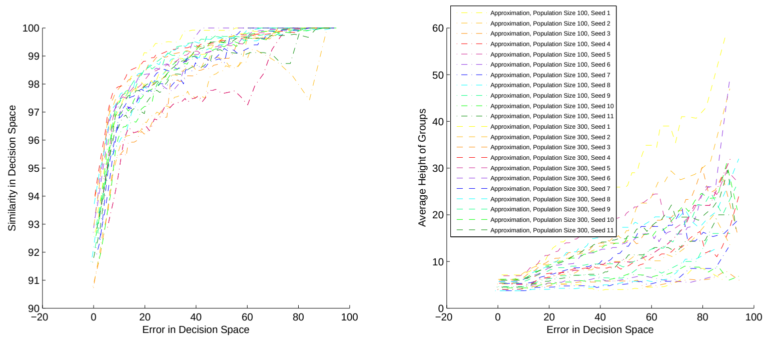


Figure 3.19: Similarity Plots for Approximations with Population Size 100 and Approximations with Population Size 300

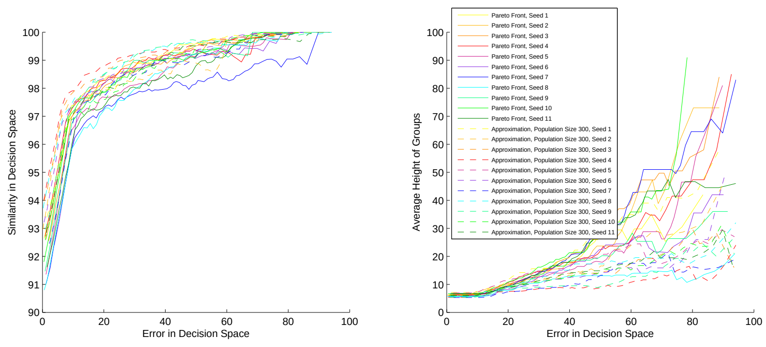


Figure 3.20: Similarity Plots for Pareto Fronts and Approximations with Population Size 300

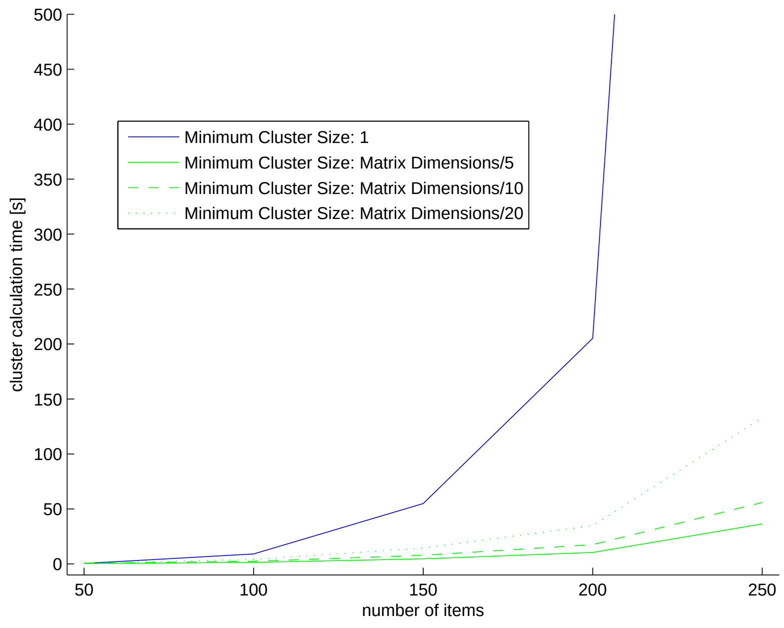


Figure 3.21: Calculation Time for Different Problem Sizes

The definition of the minimum bicluster size may be explained with an example: If the minimum bicluster size is set to a tenth of the matrix dimensions and the matrix has 300 rows and 200 columns, then biclusters which have less than 30 rows and less than 20 columns are not split again.

It can be seen that for a minimum bicluster size of 1, the calculation time increases exponentially. To alleviate this problem, the minimum bicluster size has to be increased considerably.

### 3.3.8 Application: EXPO

EXPO is a tool to optimize the architecture of packet processing devices on the system level, as described in [16]. An example scenario consists of 5 flows, 25 tasks and 8 resources, as shown in Figure 3.22[4]. Each flow consists of several tasks. The goal is to map the tasks to resources as cheaply and as efficiently as possible. Each resource has an associated cost and delay, and each task-resource pair has an associated computational demand.

After 300 generations of IBEA, the population contains 33 different solutions. All 33 solutions contain the DSP and have mapped 3 tasks (voice encoding and decoding, IPmodify) to this DSP. The two biggest subgroups both contain 7 solutions. In the first subgroup, the cipher is selected and 9 more tasks are mapped to the DSP, whereas 4 tasks are mapped to the cipher (see Figure 3.23).

[4]Figure 6 in [16]

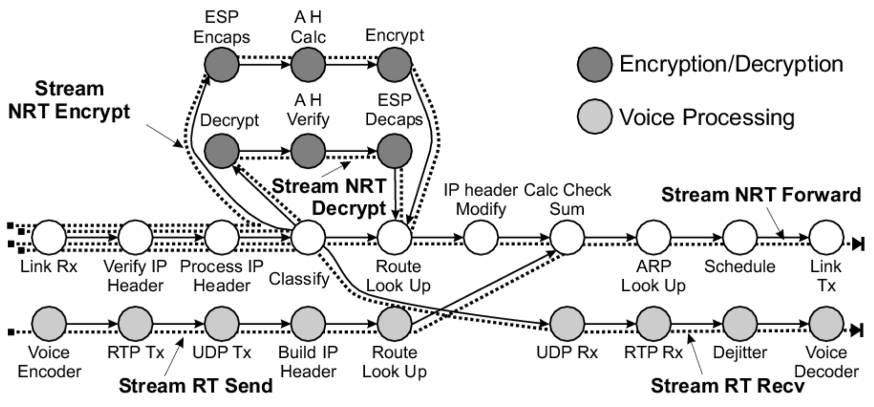


Figure 3.22: Task Graph for a Network Processor

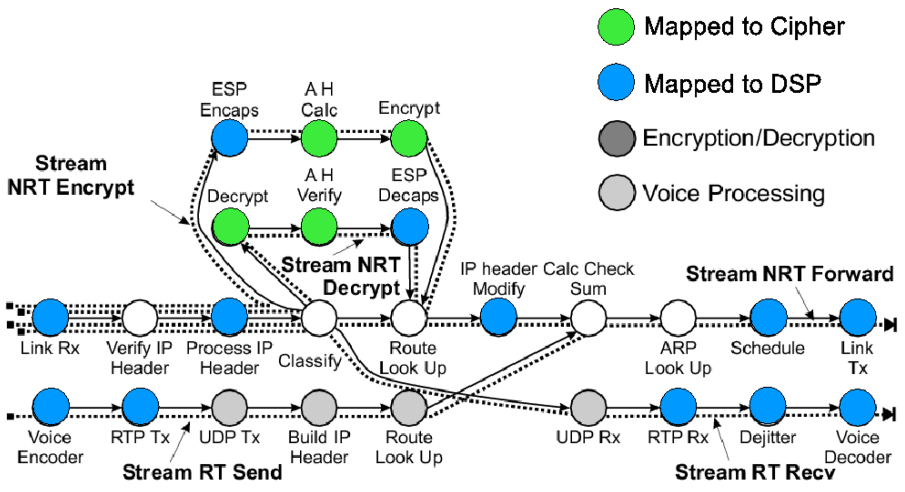


Figure 3.23: Task Graph with Mapped Tasks in First Subgroup

In the second subgroup, all 22 tasks are mapped to the DSP. The objective space values of this second subgroup are at one extreme: very cheap, but quite slow.

This already gives a good indication on what good design choices would be. Obviously, selecting the DSP generally is a good choice. If a cheap processor is desired, no other resources should be selected. If the processor is desired to be a bit faster, a Cipher may additionally be selected.

## 3.4 Summary

In this section, the various methods developed in the last chapter were tested and compared both with artificial test matrices and with real Pareto-optimal fronts with different numbers of items. Out of all of these methods, the Hartigan algorithm with Jaccard-sorting and Hartigan-splitting was chosen. This selected method was then tested further to find out about the characteristics of representation reductions which can be achieved with the developed approach.

When the selected method was applied to random matrices, it was found that the mean similarities of all groups are much smaller than those of Pareto-optimal fronts, which leads to the conclusion that Pareto-optimal fronts actually are highly structured and these structures can be used to identify modules and to groups solutions with similar modules.

To find out about the influence of the problem itself on the mean similarity of all groups, knapsack problems with a constant number of items but different seeds were tested. The results showed that the mean similarities actually depend quite strongly on the problem structure itself, which somewhat relativizes the results found for the test problems with different items.

When the groupings which were computed on the approximation of the Pareto-optimal front were compared to those which were computed on the Pareto-optimal fronts themselves, it was found that the approximations yield slightly worse results. This means that although the average group size is slightly smaller[5] the mean similarities of the groups are comparable to those of the Pareto-optimal fronts. It was also found that larger populations yield slightly better results than small populations.

Finally, the method was applied to a real-world problem, namely the network processor design of the EXPO tool. As a result some good indications could be found on what good design choices are for this network processor design.

[5]Smaller groups should yield higher similarities, as the similarity is based on the average pairwise Hamming distance of the solutions in the group.

# Chapter 4

# Online Scenario

As has been seen in the results of the last chapter, the methods developed in the offline scenario work quite well. Therefore in this chapter, they are applied to the evolutionary algorithm at runtime to speed up the optimization in order to reach the Pareto-optimal front faster. This speed up should be achieved by a reduction of the search space.

First, the online problem is defined. Then, approaches which have been taken to optimize the speed of the optimization are described.

## 4.1 Goal

When optimizing complex systems, there are many decision variables which lead to a large search space. This leads to the problem that when considering complex systems, it is difficult to reach the Pareto-optimal front in reasonable time.

In a new approach developed in this thesis, the optimization of complex systems should be sped up. This speed up shall be achieved by a reduction of the search space, which has to be done automatically and online during the evolutionary algorithm.

The underlying assumption is that some characteristics of the Pareto-optimal front already appear in the population. If such characteristics can be identified, it should be possible to reach the Pareto-optimal front in less time.

### 4.1.1 Definition of Online Problem

In an evolutionary algorithm, each decision variable is represented by one bit which is variated independently of other decision variables. In this thesis, this is called the *ordinary representation.* The approach proposed in this thesis introduces the new concept of decision variable groups. These groups consist of decision variables that ideally relate directly to each other. Such a relation exists for example if decision variable $a$ usually is selected if decision variable $b$ is selected as well. In such a case it is reasonable to merge decision variables

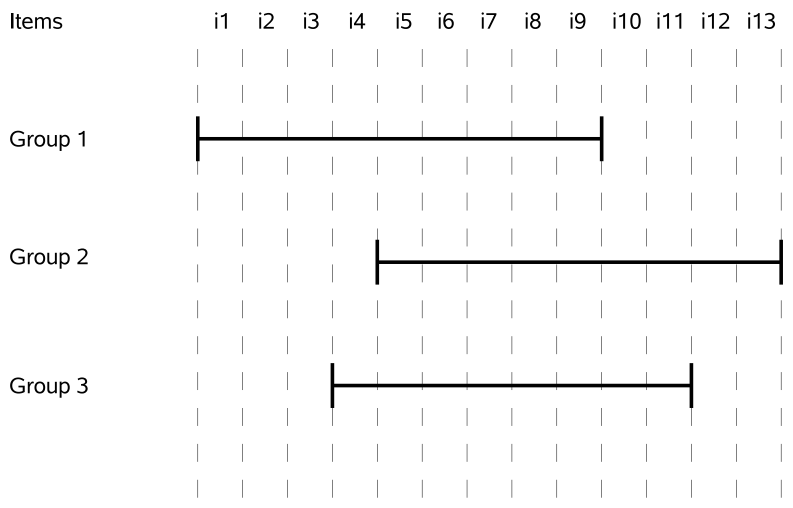


Figure 4.1: Example Groups of Decision Variables

*a* and *b* into a group and variate them together. If the number of groups is smaller than the original number of decision variables, a search space reduction has been achieved. The goal is to produce better offspring and therefore to converge more quickly towards the Pareto-optimal front.

In the adapted evolutionary algorithm, it shall be possible to switch back and forth between the ordinary representation, in which only individual decision variables are variated, and the reduced representation, in which groups of decision variables are variated. To find the groups that will be used in the reduced representation, the selected biclustering algorithm developed in the previous chapter, namely the Hartigan algorithm with Jaccard sorting and Hartigan splitting, is applied to the population. Then, the modules which are found by this algorithm are used to build groups of decision variables. Some of these groups are then selected to build the new reduced representation. This approach differs from the offline scenario approach, as in the offline scenario, the modules were used to group solutions. But in the online scenario, they are used to group decision variables instead.

Analogously to the best level cut problem (see Section 2.6), it has to be decided which of these groups should be selected to represent the population. It is possible that some groups overlap, as depicted in Figure 4.1. In this case it is not clear what subset of groups will best represent the matrix.

The biggest drawback of overlapping groups is that decision variables do not appear in the offspring with the same probability anymore. As the variator has no information about the degree of overlapping of groups, it variates the groups independently of each other. Therefore, the probability for a certain decision variable to be selected increases with the number of groups that contain this decision variable. For a fair variation process, it is therefore desirable to keep the group overlaps to a minimum. This may be achieved by requiring a tree structure on the x-axis for the Hartigan algorithm that will be used for biclustering. With such a tree structure, groups can be chosen such that they cover all decision variables, but still do not overlap.

Finally, the subset of selected decision variable groups will be coded as individual bits in the reduced representation. Decision variables that do not fit in any groups are still varied individually, like in the ordinary representation. To change back from the reduced representation to the ordinary representation, all decision variables of the selected groups in a solution are set to one, using a OR relation.

## 4.2 PISA Adaption

PISA, as described in [17], is a tool which provides an interface for optimizing a problem with arbitrary variators and selectors. It uses a file-based communication between variator and selector, where for each individual only an identity number and the corresponding objective space values are transmitted. By adapting the already existing PISA knapsack variator, tests can be run with any selector.

### 4.2.1 Integration of the Representation Change

First, the decision when and based on what individuals the representation change should be performed has to be made. The process sequence of the evolutionary algorithm is shown in Figure 4.2

As the individuals on which the biclusters are calculated should be as close to the Pareto-optimal front as possible, but still have good diversity, it was decided to do the biclustering based on the archive, after the $\mu$ selected individuals have been added. As only the variator knows the representation of the individuals in the decision space, the representation change will be calculated in the variator, shortly before the variation takes place.

To keep things simple, the representation changes after a predefined number of generations. In each generation, the variator checks whether a representation change has to be performed. As the communication between variator and selector only consists of objective space values, a representation change has no influence on the selector.

In a later stage it could be considered to do the representation change after a variable number of generations, based on the evolution of the population.

The underlying knapsack variator was taken from the PISA framework. It is able to do one-point and uniform crossover as well as one-bit and uniform mutation, both with adjustable probabilities. It assumes that $\mu$ is equal to $\lambda$. When producing infeasible individuals, their objective space values are calculated by summing up the profits of the items until the capacity bound of the

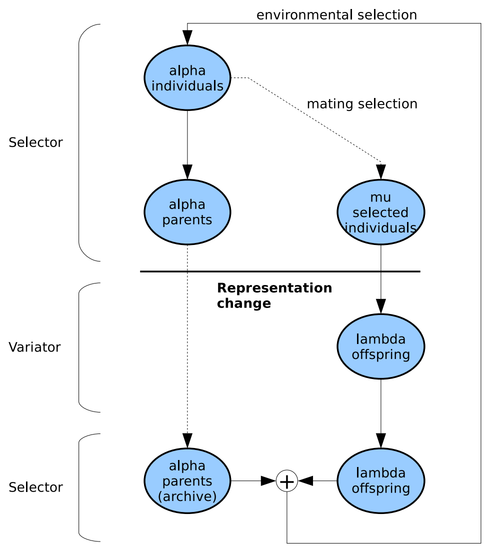


Figure 4.2: Process Sequence of Evolutionary Algorithm

first knapsack is reached. To this end, the items are sorted according to their profit/weight ratios.

As a consequence of this greedy calculation of the objective space values, most solutions in the population are infeasible. Infeasible solutions in this case imply that the solutions contain many items which do not contribute to the objective space value. As biclustering on infeasible solutions does not make sense, the calculation of the objective space values was adapted as described in Section 4.3.8.

### 4.2.2 Performance Assessment Toolbox

To evaluate a population and to calculate how fast an algorithm converges towards the Pareto-optimal front, the performance assessment toolbox[1] is used. This toolbox contains several indicators to evaluate a population, one of which is the hypervolume indicator, which is used in this thesis. As PISA only optimizes minimization problems, the knapsack problem has to be adapted. This is done by subtracting the actual profit of a solution from the total profit in the corresponding objective space dimension. In a minimization problem, the hypervolume of the population decreases in the course of the optimization.

The main program of the performance assessment toolbox is the monitor. This program coordinates the interaction between the variator and the selector. It outputs detailed population information for each generation.

[1]A detailed explanation of the function of the performance assessment toolbox can be found in [18]. Implementation issues may be found in Appendix C.2

After the variator and the selector are stopped, this information is used to calculate different indicator values to assess the performance of this particular variator/selector pair. Before the calculation of the indicator values, all objective values of the populations are normalized to the interval $[1, 2]$. Obviously, for such a normalization the maximum and minimum objective values have to be known. If, like in this thesis, these values are not known in advance, standardized values have to be used. In this thesis, the lower bound was chosen to be zero, which is the case if no items are selected. The upper bound was chosen to be $10^5$, which is the maximum profit for 1000 items, if the profit for each item ranges between 10 and 100.

## 4.3 Approaches

This section covers all approaches that were developed. These approaches range from the problem generation and the application of the biclustering algorithm to the selection of biclusters.

### 4.3.1 Generation of the Problem

Initially, the default knapsack implementation in PISA was used to generate the problem. Such a problem is defined by the profits and weights of each item and by the chosen capacity bound of the knapsacks.

The initial knapsack implementation sets the number of knapsacks equal to the number of dimensions in the objective space. Every item has different profit/weight pairs for each knapsack. The capacity bounds of the knapsacks are chosen according to the total weight of all items in the respective dimension.

A simplified approach suggested in this thesis calculates the profits similarly to the knapsack implementation of PISA, but instead of assigning each item different weights for the different dimension, they are assigned one global weight which is used for all knapsacks. In the simplest case, all items are assigned the same global weight. The capacity bound in this case is defined in terms of number of items that fit into the knapsacks. The goal is to find a subset of $n$ items that maximize the profits in all knapsacks, where $n$ is the capacity bound of each knapsack.

Up to now, the profits of the items are generated randomly. This leads to the problem that usually, a subset of these items is not dominated by any other items, whereas another subset dominates none of the remaining items. In the Pareto-optimal front as well as in approximations thereof, these items are selected in all or none of the solutions, respectively.

In real world applications such as the network processor optimization, the set of items usually only consists of indifferent items, as, for example, it does not make much sense to include a CPU which is both slower and more expensive than another CPU into the optimization.

To maintain conformity with the offline scenario, the online scenario will also make use of the initial knapsack implementation. But if a working solutions for optimization speed up is found, it should be tested on the two adapted implementations as well to learn more about its functionality.

### 4.3.2 Timing of the Representation Change

The adapted EA as proposed in this thesis is alternating between ordinary representation and reduced representation. The only variables that have to be decided upon are the generation intervals during which ordinary and reduced representation are used.

In the original algorithm with no reduced representation, the reduced representation interval simply is zero.

For a randomly initialized matrix, starting with the reduced representation makes no sense, as the decision variable groups should be based on the structure of the population.

In the simplest case, an interval of $t_g$ generations is defined. Starting with ordinary representation, the representation is changed after each $t_g$ generations.

In a next step, the intervals for ordinary representation and reduced representation could be chosen separately. The intervals could also vary with the actual generation number.

Ultimately, the intervals may be calculated online, depending on how well a population progresses.

In this thesis, only the simplest case is considered with $t_g = 50$.

### 4.3.3 Keeping Columns Fixed

As described in Section 4.3.1, there are two special subsets of items in the initial problem generation. The first one consists only of items that are not dominated by any items, whereas the second one consists only of items that do not dominate any other items. This behavior is due to the cloud shaped distribution of the items of the knapsack problem, as shown in Figure 4.3. The items of these subsets are selected in all or none of the solutions of the Pareto-optimal front. The same is true for approximations of the Pareto-optimal front, where this behavior already shows after relatively few generations.

The simplest way to speed up the approximation would therefore be to represent all items separately in the reduced representation, like in the ordinary representation, but at the same time to keep those items which occur in all ore none of the solutions fixed to 1 or 0, respectively.

In all reduced representation approaches discussed in the next sections, these columns are kept fixed. The goal is to speed up the approximation even more by varying groups of decision variables instead of individual decision variables for the remaining columns.

### 4.3.4 Types of Biclusters

In the offline scenario, two biclustering algorithms, have been used: The Hartigan and the Bimax algorithm. Advantages and drawbacks of these two algorithms are as follows:

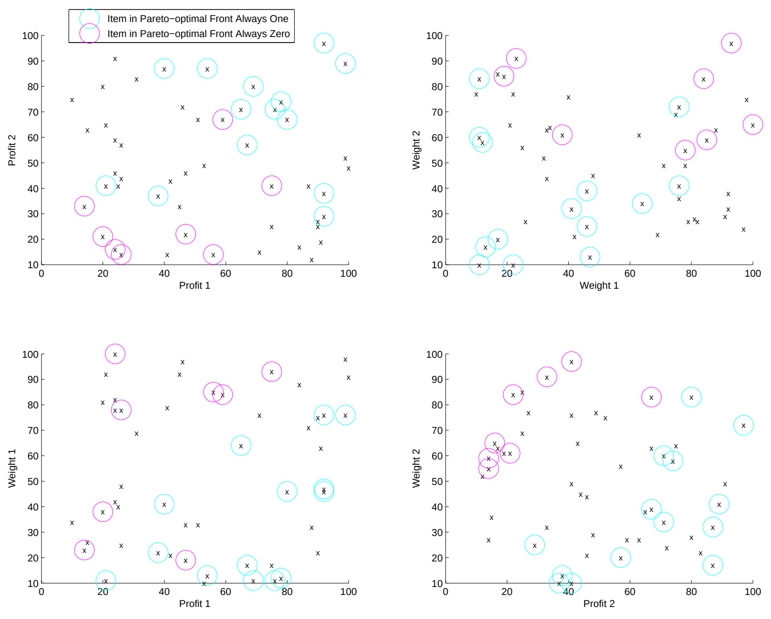


Figure 4.3: Item Distribution for Knapsack Problem with 50 Items

### Hartigan Algorithm

Many variations of the Hartigan algorithm exist. First, there are three different sorting algorithms (see Section 2.4.1) and three different split measures (see Section 2.4.2).

Furthermore, the Hartigan algorithm may require a tree structure on one or both axes. In order to alleviate the problem of overlapping groups, a tree structure of the partition is requested on the items axis.

If the derived algorithm is used, where sorting is allowed before each split, better splits may be found. On the other hand, it is not possible anymore to obtain a tree structure on the items axis.

### Bimax Algorithm

For Bimax, on the other hand, no variations exist. It is not possible to obtain a tree structure on the items axis, hence the increasing and decreasing group methods developed in the offline scenario will have to be used.

The number of biclusters found by Bimax increases exponentially with the matrix size. Therefore, the performance of the Bimax algorithm strongly depends on how many columns will be kept fixed and thus will not be used for the actual biclustering.

### 4.3.5 Identifying non-Dominated Individuals

A parameter was introduced which defines whether all solutions or only those which are not dominated by any other individuals shall be considered for the biclustering.

### 4.3.6 Identifying Unique Individuals

Similarly, another parameter was introduced which defines whether all or only unique individuals shall be considered for the biclustering. Unique in this case means that two solutions have the same decision space representation.

### 4.3.7 Choosing Biclusters

Finally, it makes sense to only consider a subset of all the decision variable groups which are found in the biclustering. The selection is based on information in the decision space. Three versions were implemented, the first of which is mandatory. For the last two, threshold variables exist that define the strength of the criterion.

#### Based on Tree Structure

To minimize bicluster overlaps (see Section 4.1.1), a tree structure on the decision variable axis is required for all biclustering algorithms. Groups of decision variables which completely overlap with other groups and therefore may be composed of other groups are discarded.

#### Based on the Approximation Population

This selection criterion calculates for each decision variable group the percentage of solutions which either contain all or no items of this group and may therefore be represented by this group. With the threshold parameter, a minimum percentage may be chosen.

#### Based on the Pareto-optimal Front

The same can be done for the solutions of the Pareto-optimal front, if the Pareto-optimal front is known. The threshold parameter again defines the minimum percentage of Pareto-optimal individuals which need to have either all or no decision variables selected in the corresponding group.

### 4.3.8 Representation of Infeasible Solutions

In the original knapsack algorithm implemented in PISA, the objective space values of infeasible solutions are calculated based on a ranking of items. In this ranking, items are sorted according to their profit/weight ratios. If the sum of weights of a solution excesses the capacity bound of at least one knapsack, the corresponding objective space values are calculated as the sum of profits of a subset of items that actually fits in all knapsacks. The subset is chosen according to the ranking of items.

In the original algorithm, this method ensures the diversity of the population. But it also leads to the fact that the population mostly consists of infeasible solutions, many of them mapping on the same objective space value. The overall result is that solutions contain more items than they would if infeasible solution were punished. This leads to the problem that there are more items which are contained in all solutions which in turn leads to wrong biclusters and groups and wrong fixed columns.

One way to circumvent this problem is to only consider items that actually fit into the knapsacks for biclustering. But then, a lot of different infeasible solutions are mapped on a small set of feasible solutions, which does not work for biclustering, because biclustering relies on a reasonably sized set of different solutions.

Therefore, it was decided to adapt the evaluation algorithm such that infeasible solutions are punished. For a maximization problem, the objective space values are defined as the sum of profits in case of a feasible solution, and as the negative sum of weights in case of a infeasible solution. For a minimization problem, the objective space value of a solution is defined as the sum of profits of all items minus the sum of profits of the items in the solution in case of a feasible solution, and the sum of profits of all items plus the sum of weights of the items contained in the solution for infeasible solutions.

## 4.4 Summary

In this chapter, the methods developed in the offline scenario have been adapted to be applied to evolutionary algorithm at runtime to achieve a reduction of the search space and therefore a speed up of the optimization.

A biclustering algorithm is used to identify modules. These modules are used to generate groups of similar decision variables. These decision variable groups are then coded as individual bits which make up the new reduced representation. To actually achieve a search space reduction, the number of decision variable groups has to be smaller than the original number of decision variables. Therefore, several methods were developed which select a subset of all found decision variable groups.

In the next chapter, this adapted algorithm will be applied to the optimization of the knapsack problem at runtime.

# Chapter 5

# Online Scenario Results

In this chapter, the different methods for search space reduction during algorithm execution, as defined in the last Chapter, will be tested. To compare the methods, only the hypervolume indicator of the performance assessment toolbox will be used (see Section 4.2.2). As this is a minimization problem, a smaller hypervolume is better. To assure a certain degree of representativeness, the resulting hypervolume curves will always be averaged over 11 problems with seeds 1 to 11.

The same parameters which were used in the offline scenario will be used (see Section 3.1).

## 5.1 Reference Curves

The reference curve is the curve against which all test results will be compared. In this thesis, two reference curves are used.

The first reference curve is given by the hypervolume curve of a state-of-the-art evolutionary algorithm. In this thesis, this algorithm was chosen to be IBEA. The parameters which are used for the IBEA implementation of PISA are shown in Table 5.1. General parameters are given in Table 5.2, where the representation interval denotes the number of generations after which a representation change from ordinary to reduced representation or vice versa is executed. The algorithm always starts in the ordinary representation. The total number of generations is chosen such that the algorithm stops in the middle of the fourth ordinary representation interval. It has also to be noted that the population on which the biclusters are calculated contains $\alpha$ parents and $\mu$ selected offspring.

As described in Section 4.3.3, the easiest way to speed up the approximation is to keep those items, which are either selected in all or none of the solutions of the population, fixed during the search space reduction interval. This method provides a simple improvement over the initial evolutionary algorithm, without yet making use of biclustering. The resulting hypervolume curve will be used as the second reference curve.

The two reference curves, averaged over 11 problem instances, are shown in Figure 5.1

| Parameter | Value |
|---|---|
| seed | 11 |
| tournament | 2 |
| indicator | 0 |
| kappa | 0.05 |
| rho | 1.1 |

Table 5.1: IBEA Parameters

| Parameter | Value |
|---|---|
| Capacity of Knapsack | 0.4 * Sum of Weights |
| Number of Items | 300 |
| Population Size: $\alpha$, $\mu$, $\lambda$ | 200 |
| Number of Generations | 325 |
| Representation Intervals | 50 Generations |

Table 5.2: General Online Parameters

## 5.2 Proof of Concept

Before starting the tests, it has to be shown that by using decision variable groups, it actually is possible to converge faster to the minimum than with the original IBEA algorithm. To this end, the decision variable groups are calculated on the Pareto-optimal front. To initialize the population, a subset of these groups is randomly selected for each solution. The optimization then starts directly in the reduced representation.

The results of this test are shown in Figure 5.2. It can be seen that the population which has knowledge of the Pareto-optimal front converges faster towards the minimum. This is mainly due to the fact that an initialization based on decision variable groups of the Pareto-optimal front is better than the random initialization of the original IBEA algorithm.

## 5.3 Tests

In this section, tests for all approaches described in Chapter 4 are executed. In the adapted algorithms, the representation of the individuals alternates between ordinary representation and reduced representation, starting with the ordinary representation. For reasons of simplicity, the intervals for both representations is set to $t_g = 50$.

In these test runs, only one biclustering algorithm is considered. Similar to the offline problem, the Hartigan algorithm with Jaccard sorting and Hartigan split measure is chosen. In contrast to the offline problem, the tree structure is required on the x-axis instead of the y-axis, as illustrated in Section 4.3.7.

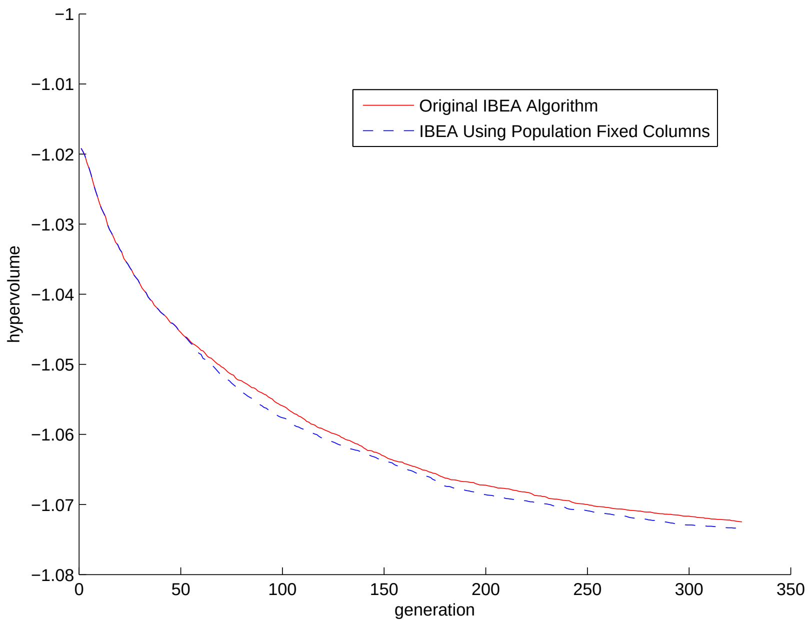


Figure 5.1: Reference Curves

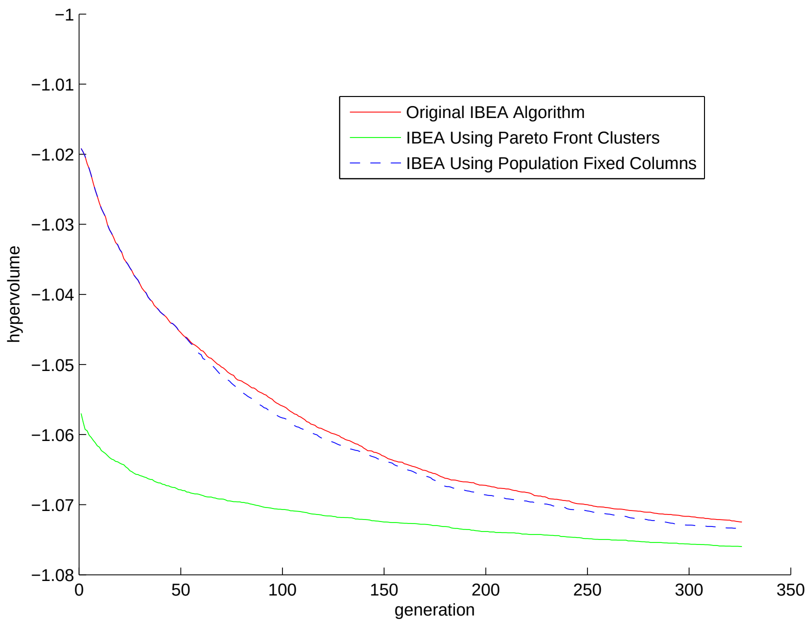


Figure 5.2: Proof of Concept: Faster Convergence to Optimum with Pareto Front Knowledge

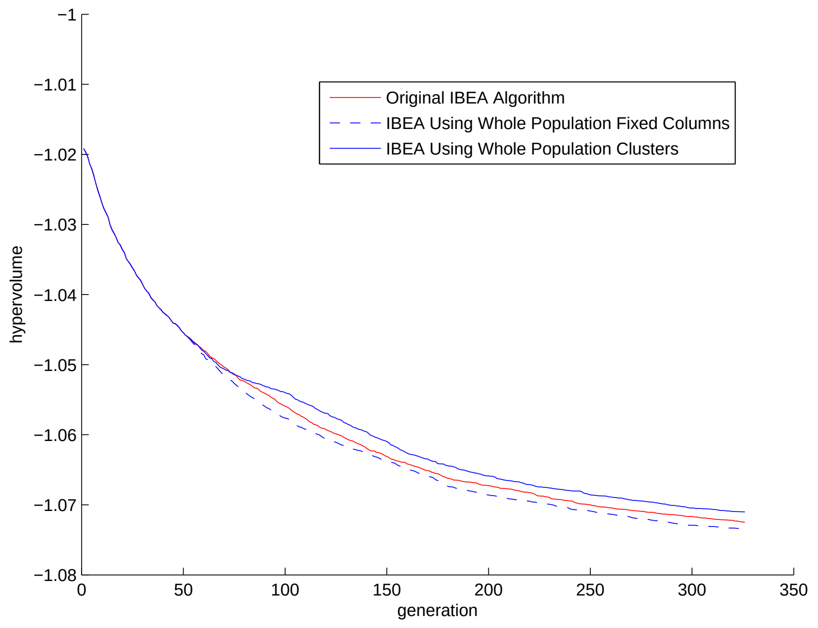


Figure 5.3: Search Space Reduction with Biclustering on Whole Population

### 5.3.1 Biclustering on Whole Population

In the simplest case, the biclustering algorithm is applied to the whole population after 50 generations, with the exception that columns containing only ones or only zeros are kept fixed. All resulting groups of decision variables, except those which may be completely composed of other groups, are selected. These groups are then varied for the following 50 generations, after which the population changes back to the ordinary representation. This cycle is repeated until the maximum number of generations is reached.

The results of this test run are shown in Figure 5.3. While for the first half of the first reduced representation interval (generations 50 to 75) the hypervolume curve follows the reference curve of the original IBEA algorithm, it deteriorates considerably in the second half. The rest of the curve maintains a constant distance to the reference curves, which means that they converge towards the minimum with similar speed.

### 5.3.2 Non-Dominated Individuals only

In a second step, the biclustering algorithm is applied only to the non-dominated individuals of the population, the results of which are shown in Figure 5.4. These results are still worse than when using the whole population for biclustering.

The number of non-dominated individuals increases with the number of generations, as there are more non-dominated individuals if the population is close to

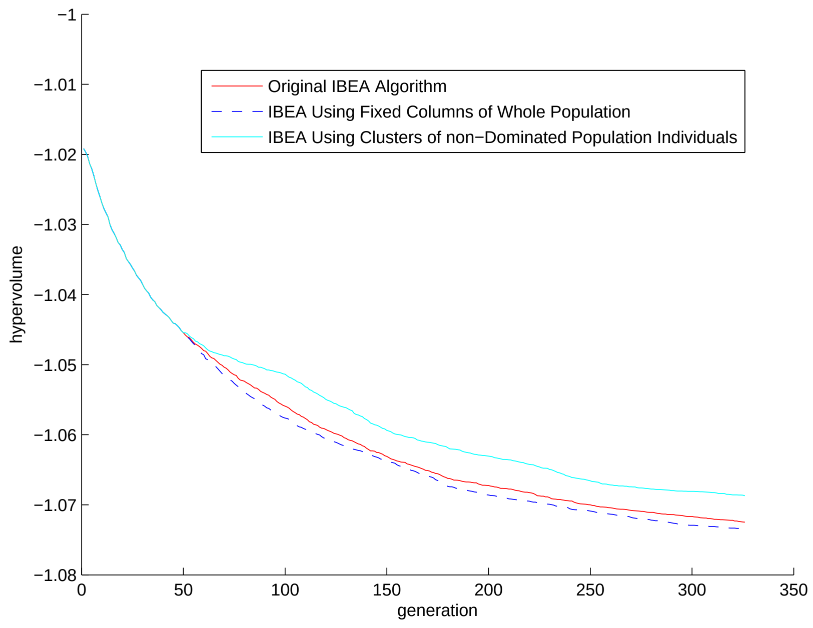


Figure 5.4: Search Space Reduction with Biclustering on non-Dominated Individuals Only

the Pareto-optimal front. Usually, these non-dominated individuals constitute about 5% to 25% of the whole population. For a population size of 400 (200 parents and 200 selected items), 5% is equal to 20 solutions, which obviously is not enough to form a representative set of solutions.

### 5.3.3 Unique Individuals only

In a third step, double individuals are discarded such that the biclustering algorithm is applied only to unique individuals of the population. The results of this test run may be seen in Figure 5.5.

It has to be noted that the population contains $\alpha$ parents and $\mu$ selected individuals, where these $\mu$ selected individuals are a subset of the $\alpha$ parents. Therefore, if only unique individuals are considered, it suffices to find the unique individuals among the $\alpha$ parents. In the test runs that led to Figure 5.5, more than 95% of the parent solutions, which is 47.5% of the whole population, usually are unique solutions. This is also the reason why the curve of the unique individual biclusters is so close to the curve of the biclusters of the whole population.

As the biclusters may be calculated much faster on the unique individuals only, and the resulting curve is very close to the one which is based on biclusters of the whole population, the biclusters will only be calculated on unique individuals for the remainder of the chapter.

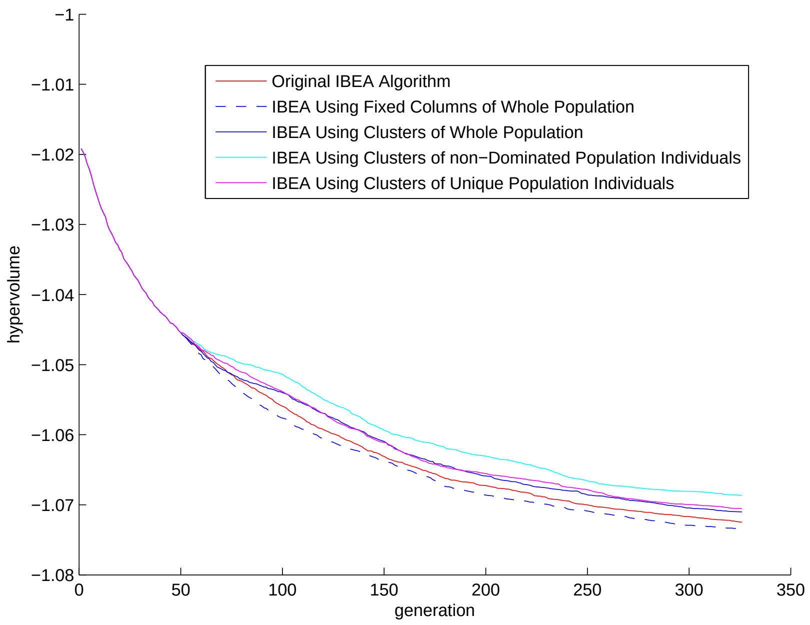


Figure 5.5: Search Space Reduction with Biclustering on Unique Individuals Only

### 5.3.4 Bicluster Selection

Another approach to reduce the number of decision variable groups further is to consider only groups that occur in a predefined number of solutions. This ensures the representativeness of the selected groups, as there has to be a certain percentage of solutions that can be represented by these groups.

A test run, in which a group is only selected if 30%, 50% or 70% of all solutions may be represented by that group, is shown in Figure 5.6. It can be seen that all resulting curves perform worse than the unmodified IBEA algorithm. Nevertheless, the results improve, the higher the predefined percentage value is. The resulting curve for a percentage value of 100% would coincide with the IBEA reference curve, as a group can only represent all solutions if these solutions all have ones in the columns that are enclosed by the group. This obviously is not possible, as all columns with only ones or only zeros have been deleted prior to biclustering. Therefore, with a percentage value of 100%, no groups would be selected, which implies that all decision variables are represented individually.

Although the selection of groups slightly improves the results, the curves perform still worse than the IBEA reference curve. It seems as if it is not possible to improve the results with the biclusters of the population. In order to confirm this statement, the biclusters will be selected based on knowledge about the Pareto-optimal front.

To this end, the groups which are found in the population are applied to the Pareto-optimal front. As before, only groups which may represent a certain

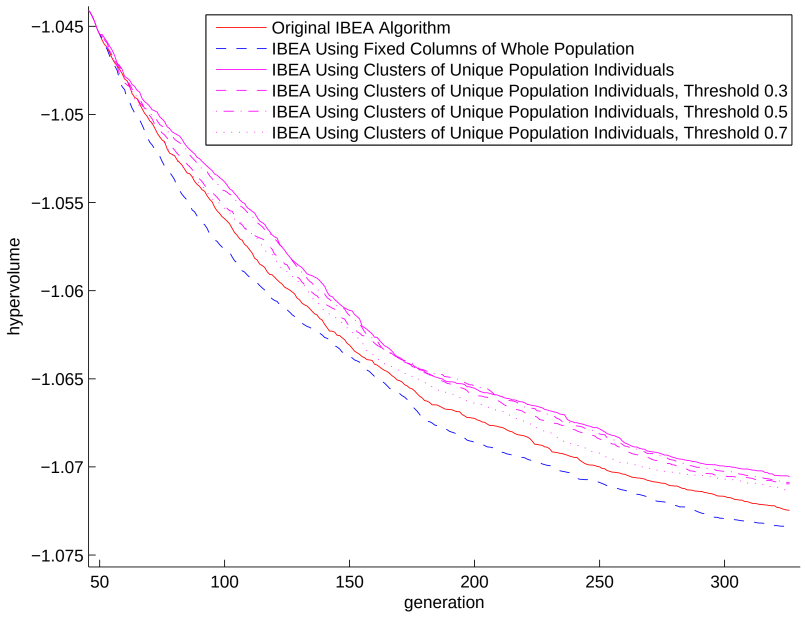


Figure 5.6: Search Space Reduction with Different Thresholds for Bicluster Selection

percentage of the Pareto-optimal solutions are selected.

A test run, in which a group is only selected if it is able to represent 20%, 60% or 100% of the Pareto-optimal solutions is shown in Figure 5.7. Even with this information about the Pareto-optimal front, the curve does not perform better than the reference curve with the fixed columns.

It seems that treating the reduced representation similarly to the ordinary representation might be too strict. For example consider a group of decision variables which is chosen because most solutions contain all of these decision variables. In this case a mutation rate, which sets the group to one and to zero with equal probability, is bound to produce many suboptimal solutions.

Section 6.3.4 on Future Work describes promising approaches which may be be pursued to tackle the persisting problems and to achieve better results.

## 5.4 Summary

In this chapter, the biclustering algorithm which has been adapted for application to the evolutionary algorithm at runtime, was tested on the knapsack problem. All results have been compared to the IBEA algorithm.

As a proof of concept, the decision variable groups are calculated on the Pareto-optimal front. The population then is initialized with these groups and optimized using the reduced representation. It could be shown that in this case, the

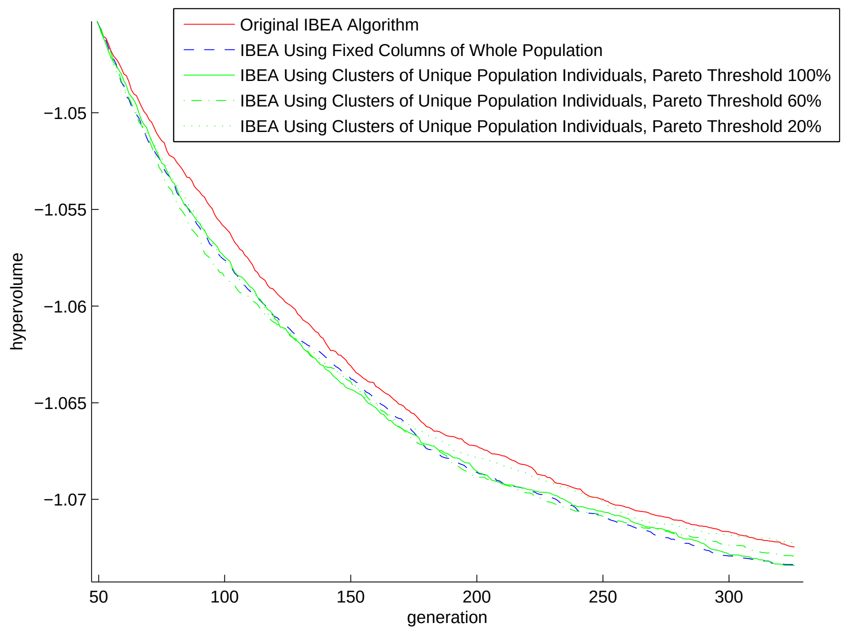


Figure 5.7: Search Space Reduction with Different Bicluster Selection Thresholds for Coincidence with Pareto-optimal Front

initialization of the population is much better than the random initialization of the original IBEA algorithm. Consequently, the population with the reduced representation reaches the Pareto-optimal front much faster than the original IBEA algorithm.

Next, the groups of decision variables were calculated on the population after a predefined number of generations. These groups are then each coded as one bit in the new reduced representation and varied accordingly. Despite the good results when using the decision variable groups of the Pareto-optimal front, optimizing the reduced representation based on the groups of the population for the next interval of generations yields worse results than the original IBEA algorithm. The reason for this might be that varying groups of decision variables in the same way as individual decision variables might be too strict.

Nevertheless, it could be shown that at least some improvements over the original IBEA algorithm could be achieved by keeping constant columns fixed for a predefined interval of generations. Constant columns are defined as decision variables, which are either selected in all or none of the solutions after a predefined number of generations.

# Chapter 6

# Conclusion

This thesis deals with two problems, both of which are related to the fact that when modeling complex systems, there are more solutions with more decision variables to consider.

The first problem, which was examined in the offline scenario, emerges after the optimization, where it is difficult to decide on a final subset of solutions of the Pareto-set approximation. It also relates to the desire to make the solutions less complex by grouping the decision variables into modules.

The second problem faced during optimization of complex problems is encountered when large populations in a large search space are optimized, as in these cases the Pareto-optimal front is only reached slowly. Therefore, an approach to speed up the optimization by reducing the search space was devised in the online scenario, using the methods developed in the offline scenario.

## 6.1 Offline Scenario

In the offline scenario, the reduction of the Pareto-set approximation consists of two stages. In the first stage, modules are found automatically using biclustering algorithms. In the second stage, solutions with similar modules are merged into groups. A module is defined as a bicluster of ones.

To find the modules, two biclustering algorithms, namely the Hartigan and the Bimax algorithm, have been selected. The modules are defined as the biclusters found by the two algorithms. The Hartigan algorithm comes in several variations, which include different sorting measures as well as different splitting measures.

In case of the Bimax algorithm which finds an exhaustive set of modules, there are no variations for the module finding stage.

In the grouping stage, on the other hand, no variations exist for the Hartigan algorithm, as the modules found by the Hartigan algorithm already are well structured and exhibit a tree structure on the solutions axis. This tree structure directly defines the grouping.

In case of the Bimax algorithm, a subset of all found modules has to be selected first. This subset is chosen such that as many ones as possible can be covered with as few biclusters as possible. As the Bimax modules are not structured, several grouping methods were developed for the Bimax algorithm. These methods include grouping according to increasing and decreasing group method, with various definition of the largeness of biclusters. This definition is used to find the order in which modules are considered to build the groups. In the increasing group method, the largest module defines the first group. The second largest module defines the next group, but with the constraint that a tree structure exists on the solutions axis, which means that the borders of the first group cannot be crossed by the second group (see Figure 4.1 for an example where such a crossing exists). This step is repeated until no new groups can be build without violating the tree structure requirement. In the decreasing group method, modules are successively discarded, starting with the smallest module. After each step, solutions, which have become identical due to the discarding of the module, are grouped.

The test cases based on which these algorithms were compared include Pareto-optimal fronts of binary two-dimensional knapsack problems with 50, 100, 150 and 200 items, respectively. The algorithms were also applied to predefined test matrices which contain obvious biclusters. The algorithms were compared in terms of the average similarity of the groups which are produced by the respective algorithm.

The results of these test runs are as follows:

- Whether the group similarities are considered in the decision space or the objective space does not make much difference.
- For the simplest predefined test case, sorting according to row and column sums is the only method which is not able to identify the correct biclusters.
- The performance of the Hartigan algorithm with sorting according to objective values strongly depends on the correlation between objective space values and decision space solutions. However, if groups which contain only neighboring solutions in objective space are desired, it is the best choice.
- In general, the results achieved by the entropy splitting measures are not better than those of other algorithms. Nevertheless, it takes much longer to compute.
- The results of the Hartigan and the Percentage splitting measure are very similar.

As a result, the Hartigan algorithm with Jaccard sorting and Hartigan split measure was chosen for the following tests, the results of which are summarized as follows:

**Random Matrices:** For Pareto-optimal fronts, the average group similarities range from 90 to 100 percent depending on the error which is considered. If the same algorithms are applied to random matrices, the average group similarities are on average 65 percent for zero error and increase linearly to 100 percent for an error of 100 percent. The average similarities therefore are much higher for Pareto-optimal fronts than for random matrices. This leads to the conclusion that Pareto-optimal fronts indeed are highly structured.

**Different Seeds:** The variation of the results from matrices with different seeds but with the same number of items is comparable to the variation of results for different numbers of items. This leads to the conclusion that the variations observed in Section 3.3.3 on different numbers of items do not actually depend on the number of items, but are caused by the matrix differences in general.

**Pareto Front vs. Approximation:** The similarities of approximations are comparable to those of the Pareto-optimal front. Nevertheless, the group sizes of the approximations are smaller and should therefore produce better results. Population sizes of 300 yield better results than population sizes of 100, as they produce slightly higher similarities with larger groups.

**Best Level Cut:** The groups which are generated by the biclustering algorithm exhibit a tree structure. In a complex system, this tree is quite large and contains many groups. To finally be presented to the user or designer, the tree has to be cut on an appropriate level. The user is only presented with groups which are located above this cut and therefore smaller groups are discarded. This can be achieved by declaring smaller modules containing only a few ones as error. Groups which are built from these small groups will then be discarded. There is a tradeoff between the considered error and the similarities of the groups, as the similarities of the groups increase if the smaller modules which cause the differing bit positions in the groups, are declared as error and are therefore not included in the similarity calculation. To find the best tradeoff automatically, a method has been developed in this thesis which performs the best level cut.

**Application on EXPO:** The methods for module finding and solution grouping were also applied to a real application, which was chosen to be the network processor design of the EXPO tool. In this design problem, a set of software tasks has to be mapped on a set of hardware resources in such a way that the resulting network processor is as cheap and as fast as possible. It was found that these methods developed in this thesis yield good indicators about what a good design decision would be, like including a DSP in any case and a Cipher in most cases and mapping several tasks on the DSP. When the groups are visualized in the objective space, it is found that the solutions at one extreme, i.e. those which are very cheap but quite slow, only contain the DSP resource.

In conclusion, the methods proposed in this thesis are able to automatically find modules in the decision space, and to group the solutions with similar modules, thereby achieving a reduction of the Pareto-set approximation.

In addition, it has been shown that there is structure both in Pareto-optimal fronts and in approximations thereof. Depending on the requirements, these structures may be highlighted using different biclustering and grouping algorithms. The methods were applied to a real network processor design (EXPO) to prove their validity. Despite the good results for smaller matrices, work still needs to be done to adapt the developed methods for use with larger matrices.

## 6.2 Online Scenario

In the online scenario, the methods developed in the offline scenario are applied to the evolutionary algorithm at runtime to speed up the optimization by reducing the search space. To this end, the modules found by the biclustering algorithms are used to find groups of decision variables. In a new reduced representation, all of these groups are coded by one bit and are varied accordingly. To find the modules, the same biclustering algorithm as in the offline scenario has been chosen.

The change to the reduced representation is done after a predefined number of generations. This change depends on several factors, such as the set of solutions on which the biclustering algorithm is applied, and the selection of a subset of groups which will finally be used for the reduced representation. To achieve a reduced representation, the number of selected groups has to be smaller than the number of the original decision variables.

The following approaches were tested:

**Usage of Groups of Pareto Front:** As a proof of concept, decision variable groups calculated from the Pareto-optimal front are used to reduce the representation in the very beginning. The population representation is initialized with these decision variable groups. This reduced representation is then varied during the whole optimization. It was found that the initialization with decision variable groups of the Pareto-optimal front is much better than the random initialization of the original evolutionary algorithm. Therefore, the Pareto-optimal front can be reached much faster.

**Keeping Constant Columns Fixed:** In the knapsack problem which is considered here, the items are distributed in a cloud-like shape. Due to this, there are several items which are either selected in all solutions, or in none of the solutions. Good results have been achieved by only keeping these constant columns fixed during a predefined interval of generations.

**Using the Decision Variable Groups of the Population:** In addition to keeping the constant columns fixed, the representation may be reduced further by varying decision variable groups instead of individual decision variables. Unfortunately, it takes more time to reach the Pareto-optimal front if such a search space reduction is used. Several approaches have been tested to improve these results, but to no avail.

The reason for this failure might be that treating the reduced representation in the same way as the normal representation may be too strict. Because as the representation changes, the mutation and recombination might have to be adapted as well to reach better results.

Apparently, using the reduced representation without modifying the variation is too strict and yields poor results. Promising approaches to tackle this problem include a completely new online approach (Appendix B) and the adaption of mutation and recombination probabilities (Section 6.3.4).

Despite these problems with varying decision variable groups, it could be shown that at least some improvements can be achieved by keeping constant columns fixed for a predefined interval of generations.

## 6.3 Future Work

### 6.3.1 Generalization to Real Valued Matrices

In this thesis, only binary decision spaces have been considered. But as most real life applications have a real-valued decision space, it makes sense to extend the methods developed in this thesis to real valued matrices. To this end, a new definition of biclusters has to be found. As the Hartigan algorithm initially was intended for use in real valued matrices, only the stop criterion, which has been adapted for this thesis, needs to be reverted to its initial definition.

As the Bimax algorithm has specifically been designed to find biclusters in binary matrices, the focus should lie on the adaption of the Hartigan algorithm.

### 6.3.2 Modification of Hartigan for Speed

For large matrices, the calculation of all biclusters found by the Hartigan algorithm becomes infeasible. Instead of simply increasing the minimum bicluster size, a new minimum bicluster size which is based on the distribution of ones in the biclusters could be developed. With such a new definition, biclusters which contain mostly zeros could be discarded before they actually contain nothing but zeros anymore. This is acceptable, as in this thesis, the focus lies on biclusters of ones.

### 6.3.3 Modification of Bimax for Speed

For Pareto-optimal fronts of problems which have more than 200 decision variables, the Bimax algorithm quickly becomes infeasible. Increasing the minimum bicluster size helps, but to obtain a reasonable number of biclusters, the minimum bicluster size needs to be increased considerably, sometimes up to a quarter of the matrix dimension sizes. Obviously with higher minimum bicluster sizes, the error increases as well.

As only a subset of all biclusters found by the Bimax algorithm is selected for grouping, it would make sense to adapt Bimax such that it recognizes biclusters which will probably not be chosen.

### 6.3.4 Modification of Online Variation

As shown in Section 5.3.4, the methods used for variation of single decision variables do not apply well to the variation of groups of decision variables. Therefore, mutation rates might have to be adapted such that they fit the given decision variable group. For example a decision variable group, which was formed because most solutions have selected all of the decision variables in this group, should have a higher probability to be set to one than to be set to zero.

To learn more about what constitutes a good variation, a closer look into the mechanisms of representation changes has to be taken. A good point to start is to compare the effect variation has on the different representations of a sample population. Different variation methods will have to be tested to eventually find the one which yields the best results for the reduced representation.

# Appendix A

# Offline GUI

During this thesis, a graphical user interface (GUI) has been developed, with which all methods that have been developed can be tested on different problem sets. In the following sections, the different components of the GUI are described.

## A.1 Menu

When using the GUI, the first step is to select a matrix. Afterwards, a biclustering algorithm may be applied to the matrix. These two steps are shown in Figure A.1.

After the biclustering algorithm has been selected, the biclusters are calculated and the groups are generated and illustrated in a first figure (see Figure A.2 for an example of a Pareto-optimal front with 50 items and biclusters calculated by the Hartigan algorithm with Jaccard sorting and Hartigan splitting). Solutions belonging to the same group are shown in the same color.

Note that this figure only shows the uppermost group level. Group levels and single groups may be selected from the menu as shown in Figure A.1.

## A.2 Error Panel

In the error panel, the desired error can be set with the slider. As soon as the error has been set, it has to be confirmed by pressing the *Set Error According to Slider*-Button. This in turn produces a structure/error plot (see Figure A.3), in which the first bicluster that is above the selected error bound is marked by a red circle. All biclusters to the right of this marked bicluster are discarded. The remaining biclusters may be shown in the decision space by pressing the *Show Clusters in Decision Space*-Button (see Figure A.4). In this figure, the ones of the matrix are marked as dark blue squares, whereas the zeros are marked as light blue squares. Ones considered as error are marked as magenta squares. Next to the matrix, the groups are shown.

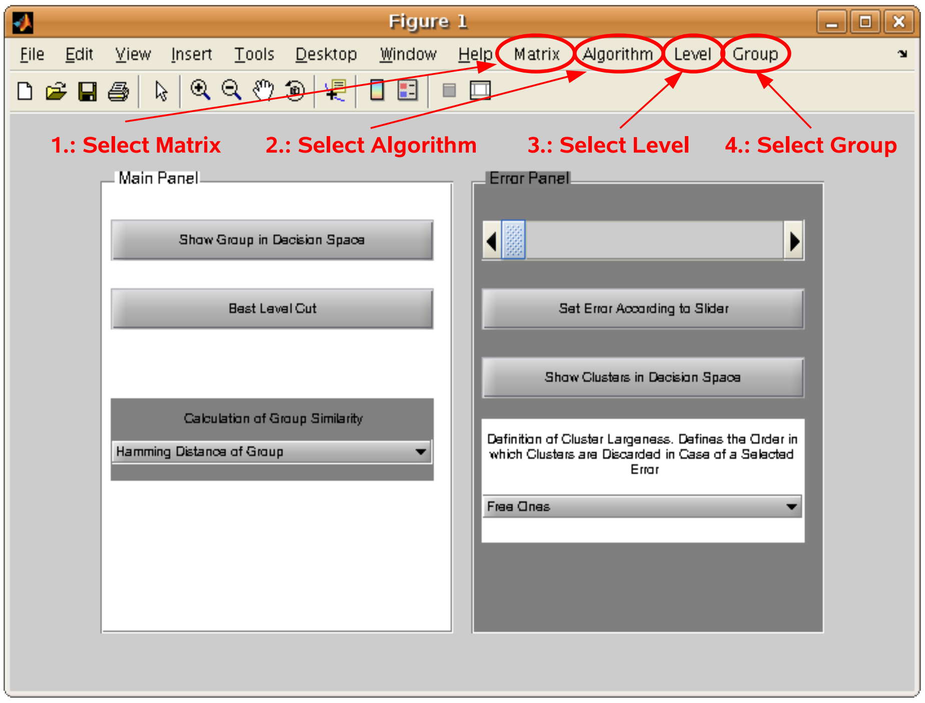


Figure 1
File Edit View Insert Tools Desktop Window Help Matrix Algorithm Level Group
1.: Select Matrix
2.: Select Algorithm
3.: Select Level
4.: Select Group
Main Panel
Show Group in Decision Space
Best Level Cut
Calculation of Group Similarity
Hamming Distance of Group
Error Panel
Set Error According to Slider
Show Clusters in Decision Space
Free Ones

Definition of Cluster Largeness. Defines the Order in which Clusters are Discarded in Case of a Selected Error

Figure A.1: Screenshot of the GUI

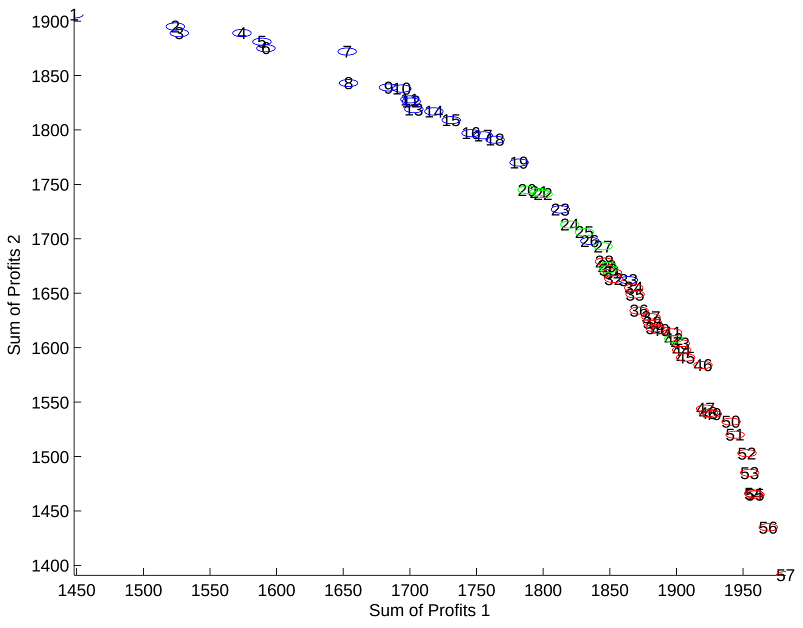


Figure A.2: First Level of Groups in Objective Space

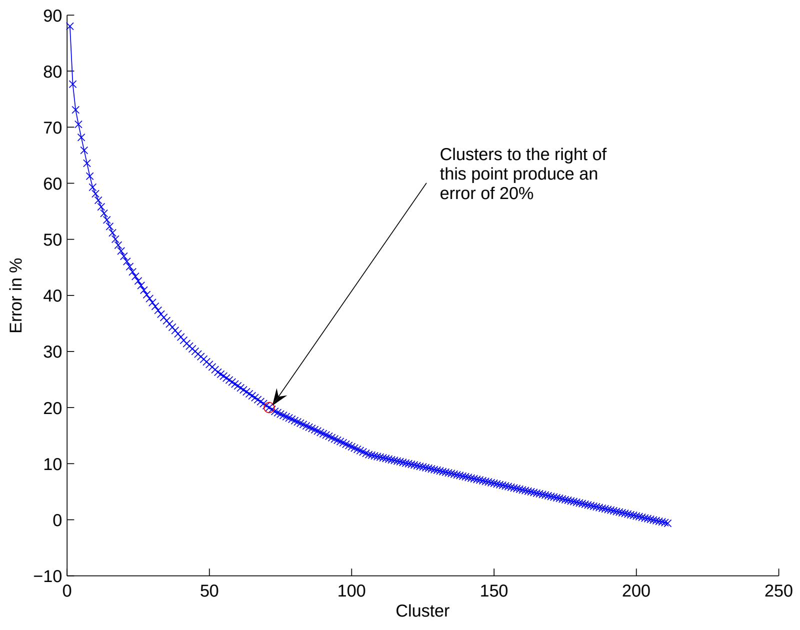


Figure A.3: Structure/Error Plot

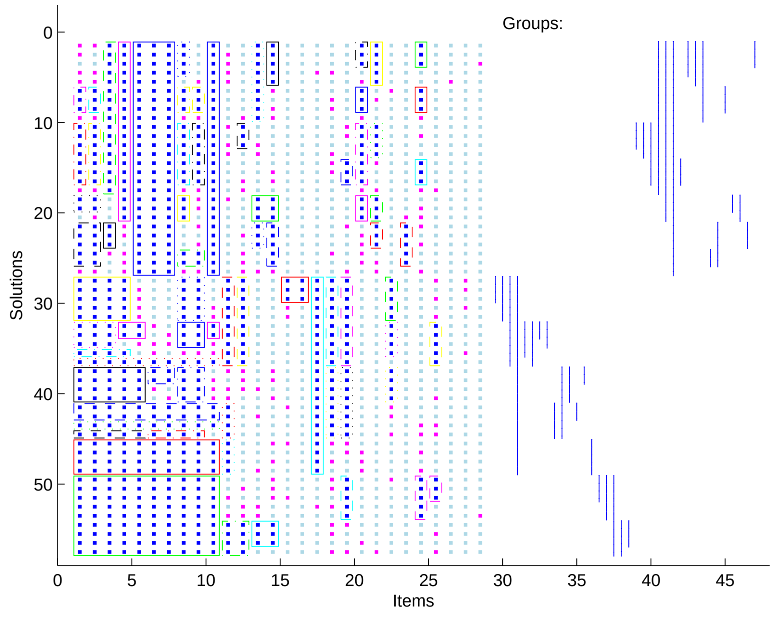


Figure A.4: Biclusters in Decision Space

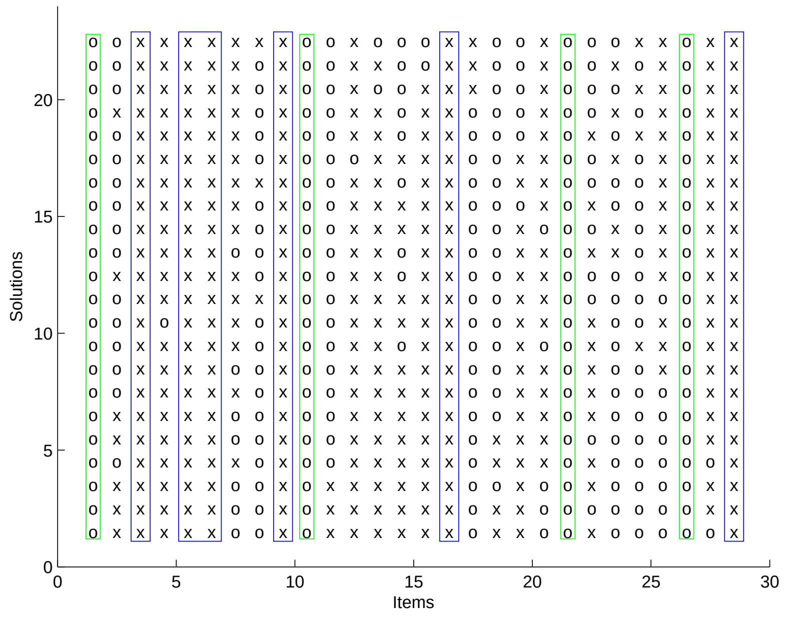


Figure A.5: Example Group in Decision Space

To increase the error, biclusters are discarded from smallest to largest. To select different definitions of the largeness of a bicluster, the desired option may be chosen from the drop down menu in the error panel.

## A.3 Main Panel

In the main panel, the solutions of a chosen group may be displayed in the decision space by pressing the *Show Decision Space*-Button. See Figure A.5 for an example, where the columns for which all solutions have the same values are marked. At the same time, the item distribution is drawn in a second plot (see Figure A.6). Again the columns with fixed values for all solutions of that group are marked.

With the *Best Level Cut*-Button, the best level cut is displayed, as shown in Figure A.7. Different similarity measures may be selected from the drop down menu. Note that for the definition of the largeness of biclusters, the value chosen from the drop down menu in the error panel is used.

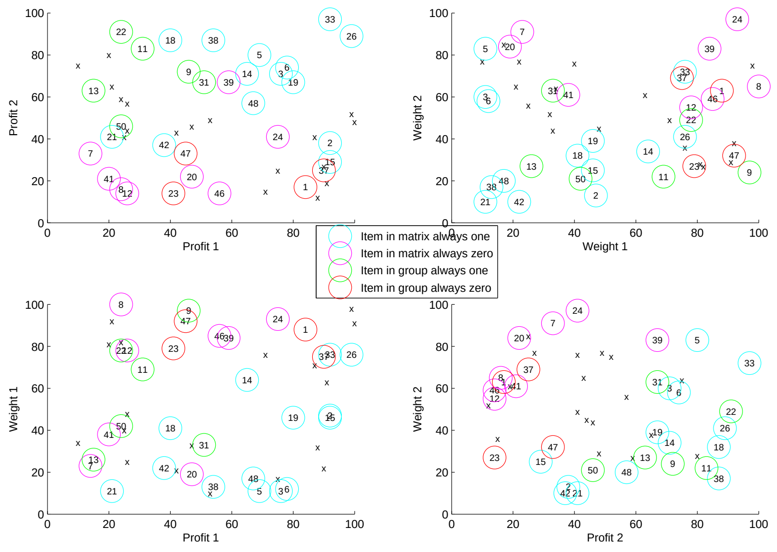


Figure A.6: Item Distribution of a Group

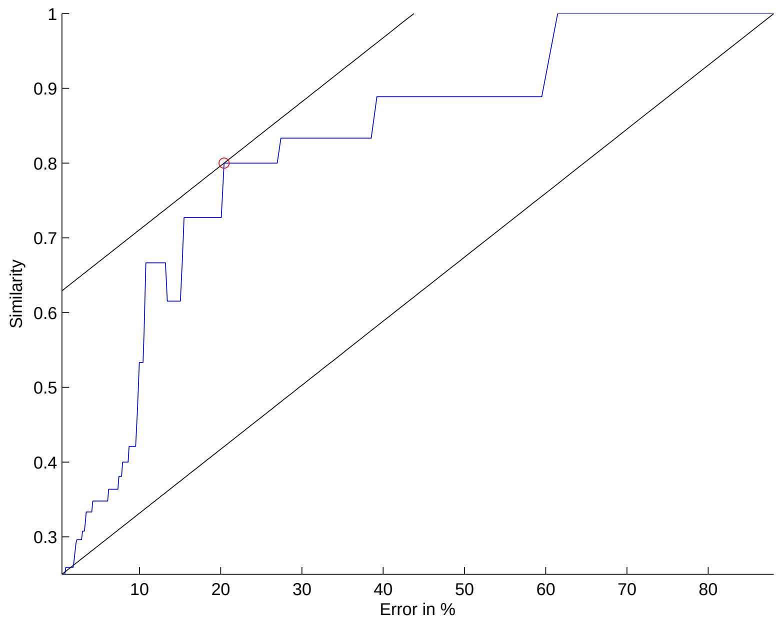


Figure A.7: Best Level Cut in Error/Similarity Plot

Appendix B

# Completely New Online Approach

Making use of decision variable groups seemingly does not yield the desired results. It is possible that some modification of the module finding and grouping methods will be found which improves the results. Nevertheless, there is a slight chance that the grouping approach proposed in this thesis simply is not appropriate to reduce the search space online.

This section formalizes the reduction problem in a very general way. Based on this formalization, a new approach to search space reduction may be developed.

Given a set of columns $S_0 = \{\{1\}, \{2\}, \dots, \{k\}\}$, the goal is to find a final set of column subsets $S_T = \{s_1, s_2, \dots, s_l\}$, where $s_i \subseteq \{1, \dots, k\}$. Each initial column selection $s_i = \{i\}$ is equal to only one column. Each column is assigned a row set $z_i \subseteq \{1, \dots, n\}$. This row set indicates in which solutions the decision variable of the concerned column is selected. To reach $S_T$, the column sets $s_i$ have to be combined. In a special case, the column sets have identical row sets and are therefore identical. In this case, these columns may be combined without information loss. In the general case where the row sets are different, a method has to be developed which defines the order in which column sets shall be combined.

Columns which are merged need to be assigned a new representation. For example if two columns have to be merged, they are coded by one bit in the new representation. It first has to be decided which settings should be used for the two columns if the bit is one or zero. The straight-forward approach would be to assume that both columns are one or both are zero if the bit representing these two columns is set to one or zero, respectively. A more general approach does not simply set the two columns to the same value, but to the two representations which best represent the relationship between the two columns.

To evaluate such representations, a quality measure of a column set $S^*$ is introduced as the sum of qualities of each representation $S_R \subseteq S^*$ of these columns. For example if the set only contains two single columns, possible representations are $\{00, 01, 10, 11\}$. The column set then is coded into one single bit by setting 1 for the best ($S_1$) and 0 for the second best ($S_2$) representation.

Again, several measures to calculate the quality of a representation exist. In the case where columns are merged, the quality may be defined as:

$$Q(S_R) = \sum_{1 \leq j \leq n} \text{Fit}(i) \cdot \text{Rating}(j)$$

W $j$ ist a row, Fit$(j)$ is the fitness of the row and Rating$(j)$ is defined as follows:

$$\text{Rating}(j) = \max\{\text{Dev}(j, j|S^* = S_1), \text{Dev}(j, j|S^* = S_2)\}$$

Here, $S_1$ is the best and $S_2$ the second best representation of the column set $S^*$. The Deviation may be defined in three ways:

$$\text{Dev}(j) = \begin{cases} \text{I}_\epsilon(j, j|s) \\ \text{Fitness difference}(j, j|s), \text{ with rest of population constant (e.g. I}_H) \\ \sum_{1 \leq i \leq d} \text{dist}(j, j|s), \text{ where } i \text{ is the dimension in objective space} \end{cases}$$

The epsilon indicator $I_\epsilon$ may either be defined as the maximum or the sum of the epsilons in all directions. $I_H$ is the hypervolume indicator.

In the case that columns $S^*$ are discarded, the bit is set to a constant $S^* = S_1$, where $S_1$ is the best representation.

Depending on the quality of the column set, it can be either discarded or merged with other columns.

Notation: $j|S^* = S_r$ means the $j$-th solution, whose bits are set as in representation $S_r$.

# Appendix C

# Online Scenario Implementation Issues

## C.1 Starting Matlab out of a C Program

In this thesis, the offline scenario has been implemented in Matlab. The only exception is Bimax, which has been taken from the SOP Homepage [11] and which is implemented in C.

For the online scenario, the knapsack implementation of the PISA framework [17] is used. This implementation is only available in C source code. As the adaption of the variator makes use of the methods developed in the offline scenario, a simple way to call Matlab functions out of a C program had to be found.

To achieve this, the Matlab Engine provided by Matlab was used. With this engine, a Matlab function $a = func(b)$; may be called easily with the following syntax:

```
#include "engine.h"
Engine* ep = engOpen("\0");
engEvalString(ep, "a = func(b);");
engClose(ep);
```

As can be seen from this example, the `engine.h` file has to be included. Furthermore, the linker has to be able to find the appropriate libraries. Therefore, the linker library path variable (`LD_LIBRARY_PATH`) has to be set before running the compiler.

## C.2 Changes to the Performance Assessment Toolbox

To be able to compare different optimization runs, the hypervolume indicator of the performance assessment toolbox [18] is used. There are several issues that require attention when using the performance assessment toolbox. The following points describe the changes which have been made to the performance assessment toolbox.

- To change the number of generations, it is not sufficient to change the `run.ksh` file. It is also necessary to adapt the `monitor_param.txt` file. Furthermore, the number of generations in the `knapsack_param.txt` file has to be larger than the actual number of generations by at least two generations. The reason for this is that the monitor should have full control over the knapsack variator, including its destruction.
- It is very important to delete all contents of the `runs/` and the `tests/` directories. The monitor saves population information for each generation in these directories, which will later be used by the indicators. The indicators use all information in these directories, as they have no possibility to identify out-of-date information.
- The monitor initializes the state of the variator and the selector to 8 and 10, respectively. This ensures that both variator and selector reset before any calculations start. Nevertheless, the variator automatically executes state 0 before entering the state machine, which should be prevented when using it with the monitor, as state 0 is executed shortly after state 8 anyway.

# Bibliography

[1] M. Preuss, B. Naujoks and G. Rudolph: *Pareto Set an EMOA Behavior for Simple Multimodal Multiobjective Functions* In: Parallel Problem Solving from Nature - PPSN IX (2006), ISBN 3-540-38990-3

[2] K. Deb and A. Srinivasan: *Innovization: Innovative Design Principles Through Optimization* (2006)

[3] Y. Cheng and G. M. Church: *Biclustering of Expression Data* (2000)

[4] J. H. Holland: *Hierarchical descriptions of universal spaces and adaptive systems* (Technical Report ORA Projects 01252 and 08226). Ann Arbor: University of Michigan (1968)

[5] J. H. Holland: *Adaptation in natural and artificial systems.* Ann Arbor: The University of Michigan Press (1975)

[6] D. E. Goldberg: *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley Publishing Company, ISBN 0-201-15767-5 (1989)

[7] D. A. Van Veldhuizen: *Multiobjective Evolutionary Algorithms: Classifications, Analyses, and New Innovations* PhD thesis, Graduate School of Engineering of the Air Force Institute of Technology, Air University, June 1999.

[8] D. E. Goldberg et al.: *Messy Genetic Algorithms: Motivation, Analysis, and First Results* In: Complex Systems, 3:493-530 (1989)

[9] D. E. Goldberg et al.: *Messy Genetic Algorithms Revisited: Studies in Mixed Size and Scale* In: Complex Systems, 4:415-444 (1990)

[10] J. A. Hartigan: *Direct Clustering of a Data Matrix*, In: Journal of the American Statistical Association, March 1972, Volume 67, Number 337

[11] A. Prelic, S. Bleuler, P. Zimmermann, A. Wille, P. Bühlmann, W. Gruissem, L. Henning, L. Thiele, E. Zitzler: *A systematic comparison and evaluation of biclustering methods for gene expression data* In: Bioinformatics, 22(9): 1122 - 1129, 2006 Implementation on http://www.tik.ee.ethz.ch/sop/bimax/

[12] S. C. Madeira and A. L. Oliveira: *Biclustering Algorithms for Biological Data Analysis: A Survey* IEEE/ACM Transactions on computational biology and bioinformatics

[13] R. Tibshirani, T. Hastie, M. Einsen, D. Ross, D. Botstein and P. Brown: *Clustering methods for the analysis of DNA microarray data.* Technical report, Stanford University (1999)

[14] G. Duffy and A. Quiroz: *A permutation based algorithm for block clustering* In: Journal of Classification, 8:65-91 (1991)

[15] C. Haubelt, S. Mostaghim, J. Teich and A. Tyagi: *Solving Hierarchical Optimization Problems Using MOEAs* In: Evolutionary Multi-Criterion Optimization (2003)

[16] L. Thiele, S. Chakraborty, M. Gries and S. Künzli: *A Framework for Evaluating Design Tradeoffs in Packet Processing Architectures* In: Proc. 39th Design Automation Conference (DAC), pp 880 - 885, 2002

[17] S. Bleuler, M. Laumanns, L. Thiele, E. Zitzler: *PISA - A Platform and Programming Language Independent Interface for Search Algorithms*

[18] C. Fonseca, J. Knowles, L. Thiele and E. Zitzler: *A Tutorial on the Performance Assessment of Stochastic Multiobjective Optimizers* Feb. 2006, Implementation on http://www.tik.ee.ethz.ch/sop/pisa/?page=assessment.php