SEMESTER THESIS

# Visualizing High-Dimensional Data

**Computer Engineering and Networks Lab (TIK)**
**Prof. Dr. Eckart Zitzler**
**Fall term 2008**

*Author:*

Jeremie Bresson

*Supervisors:*

Tamara Ulrich
Johannes Bader

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 Motivation

Finding the desired solution of a complex problem with a lot of design parameters and many objectives is not an easy task. Multiobjective search algorithms exist [11] and are able to help in this task by sorting out the best solutions. Because the objectives are often conflicting, the solution of the problem is not unique and the best that can be done is to find a set of solutions, where no solution is better that the others for all the objectives.

At the end, somebody must choose among all the suggested solutions. This decision maker is very often a human, which will make his choice based on his experience, additional knowledge he did not put in the algorithm, or just his intuition. During this task, it is very important that he comprehends the proposed set of solutions. In this task he needs a tool that provides him a good view of the set of solutions.

Displaying these solutions raises some problems:

## 1.2 Problems

### 1.2.1 High dimensional spaces

Each solution can be considered as a high dimensional point, where each coordinate represents the value of this solution for the different objectives. The search algorithms give a set of points living in a $p$-dimensional space (where $p$ is the number of objectives). In addition, there are many design parameters that construct the solu-

tions. It is also possible to interpret this construction as a vector lying in another high dimensional space.

The problem with high dimensional spaces is their representation. A screen or a sheet of paper is only a two-dimensional area. We are used to representing two-dimensional spaces with a Cartesian coordinate system in the plane. A three-dimensional space can still be represented with perspective methods. For higher dimensional spaces, the representation becomes a complex task.

The first possibility is to use high dimensional plotting methods like parallel plots, suggeted by Inselberg [3]. The advantage of this representation is that no information is lost. But these graphs are not easy to read and to understand. So this view is not fully satisfying.

A second approach is to map the high dimensional space onto a two-dimensional one. Such a mapping is called a dimension reduction. two-dimensional graphs seem natural and are easier to understand. A drawback is that the representation of a high dimensional space with only two dimensions, induces simplification and lost of information. That's why dimension reduction is not an easy task.

### 1.2.2 Large number of points

A second major problem with the multiobjective search algorithms is that they produce large solutions' sets. They are able to eliminate solutions that are worse than others for all the objectives, but as soon as a solution is better than another one for at least one of the objectives, it has to be kept. The more objectives the problem contains, the more solutions are found by the algorithm.

It is important to understand how the set of solutions is organized, which structure is behind the data. Some solutions might be similar to some others, but very different to other ones. A large number of points confuses the decision maker. A simplification of the problem is to focus only on groups of similar solutions. This enables first to compare the different groups, and in a second step to investigate only the interesting groups.

Building groups of points is called a clustering procedure. A lot of methods exist, but they need to be adapted to the set of points we are dealing with in multiple optimization problems.

## 1.3 Goals

The thesis is spitted in three main parts:

The first part deals with dimension reduction methods. Thereby the goal is to evaluate how good is the mapping from a high dimensional space on a two-dimensional one. This enables to compare two dimension reduction methods on given test sets, and helps to decide which method is more suitable.

The second part's goal is to study clustering procedures. The goodness of a clustering procedure needs to be measured. The goal is to extend a clustering procedure and to investigate the results.

The last part is related to the realization of the Java visualization tool. A design for the architecture is suggested. The Java application integrates the work done in the first two parts.

The report is organized as following. The second chapter gives some backgrounds on the multiple objective optimization. The dimension reductions techniques are studied in the third chapter and the clustering procedure in the fourth. The fifth chapter provides information on the implemented tool. The conclusion is given in the sixth chapter.

# Chapter 2

# Background

## 2.1 Multiobjective Optimization

Finding an optimal solution for a problem is a very common task. That's why optimization problems are studied and used in a wide range of domains. Finding the best solution amongst possibilities is quite an easy task if a single objective is considered. It becomes much more complex for the optimization of many different criteria. Because the several objectives are often conflicting, the result of a multiobjective search is not a single point, but a set of points. Compromises need to be made, and they increase the number of interesting solutions. Unfortunately the unique best solution for all the objectives is a utopia in most of the cases.

A famous and simple example of multiobjective optimization is the knapsack problem. A housebreaker enters a room where he can take objects. Each object has a given weight and a given value. The housebreaker can put as many items as he wants in his knapsack. He is confronted with the following problem: he wants to pack the items that constitute the lightest and the most profitable knapsack.

In other words the housebreaker has to select among all the possible objects' combinations, the one that optimizes two conflicting objectives: minimizing the weight and maximizing the profit. It is easy to see that there is not a single solution. An empty knapsack has the smallest weight, but unfortunately also the smallest value. On the other side, the knapsack containing all objects has the highest value but also the highest weight. Between these two extrema, all the knapsacks that make compromises between the two antagonist objectives are interesting.

A simple instance of the knapsack problem is presented in Figure 2.1. Three possible knapsacks constructed with these items are presented in Figure 2.2.

The knapsack problem constitutes a good basis example of multiobjective op-

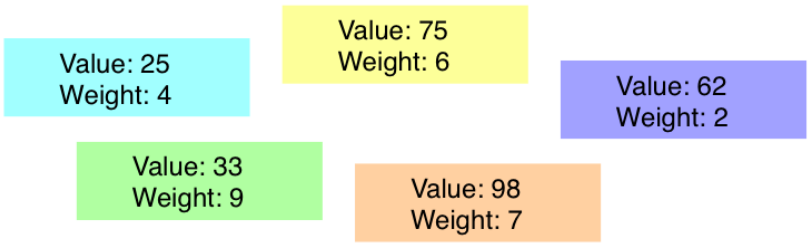


Figure 2.1: Set of all the available items

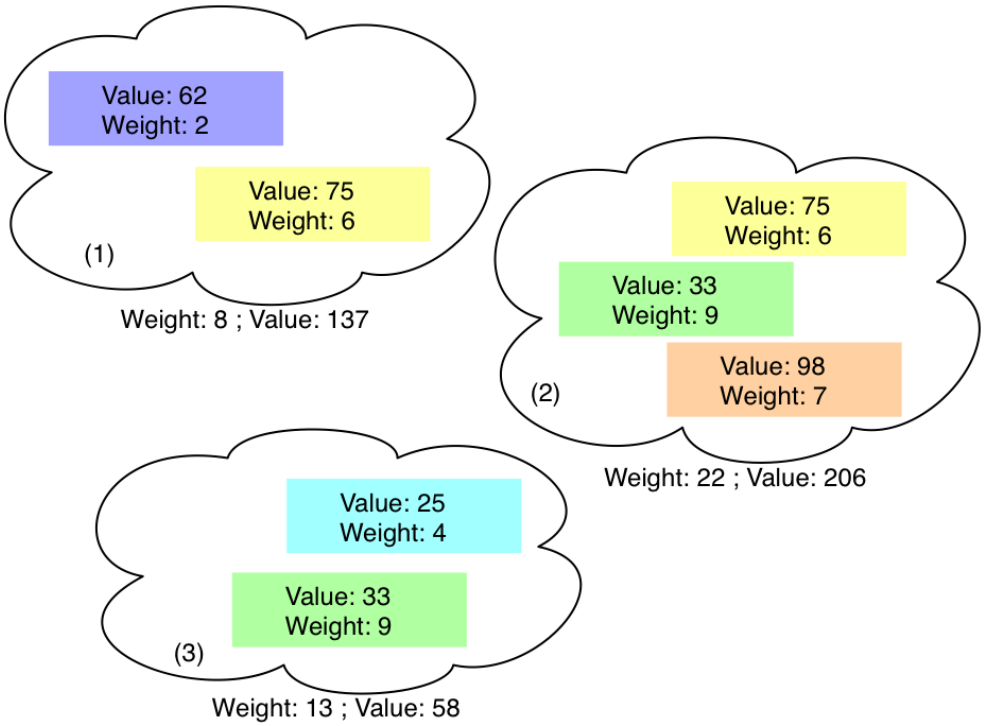


Figure 2.2: Knapsack (1) (2) and (3)

timization. In order to simulate more complex problems, this problem can be extended: new objectives are added ([11]). For each objective, it is important to know if it needs to be maximized or minimized.

## 2.2 Objective and decision space

A multiobjective problem can be modeled as a vector function $f$ that maps a tuple of $q$ parameters (the decision parameters) to a tuple of $p$ objectives.

Knowing $f$ the decision vector is sufficient to define a point (the objective vector can be calculated with the function $f$). The objective vector is very important (the value of the objectives is at the end what matters in order to evaluate a point). So at the end a point is considered as a couple of an objective vector and a decision vector.

The objective space is the vector space that contains all possible objective vectors. The decision space is the vector space that contains all possible decision vectors.

Following notation is used:

A point (i) is a tuple of an objective vector and a decision vector:

$$P^{(i)} = (O^{(i)}, D^{(i)})$$

The objective vector is in the objective space:

$$O^{(i)} = (o_1^{(i)}, o_2^{(i)}, ..., o_p^{(i)}) \in \mathbb{R}^p$$

The objective space is denoted:

$$\mathcal{O} = \mathbb{R}^p$$

The decision vector is in the decision space:

$$D^{(i)} = (d_1^{(i)}, d_2^{(i)}, d_3^{(i)} ..., d_q^{(i)}) \in \mathbb{R}^q$$

The decision space is denoted:

$$\mathcal{D} = \mathbb{R}^q$$

We can identify such spaces in our knapsack problem introduced in Section 2.1.

There are 2 objectives: weight and profit. The decision space is the tuple of all the objectives, so in our case:

$$\mathcal{O} = \mathbb{R}^+ \times \mathbb{R}^+$$

The dimension of this objective space is $p = 2$.

The decision space can be considered as following: we have 5 items. Each item can be in the knapsack (value 1) or not (value 0). There is one binary parameter for each item. The decision space is then:

$$\mathcal{D} = \{0,1\} \times \{0,1\} \times \{0,1\} \times \{0,1\} \times \{0,1\}$$

Its dimension is $q = 5$.

The table 2.1 gives the coordinates of the three knapsack presented in Figure 2.2.

| | Decision space | | | | | Objective space | |
|---|---|---|---|---|---|---|---|
| | *item in the knapsack?* | | | | | *weight* | *value* |
| | $d_1$ | $d_2$ | $d_3$ | $d_4$ | $d_5$ | $o_1$ | $o_2$ |
| Knapsack (1) | 0 | 1 | 0 | 0 | 1 | 8 | 137 |
| Knapsack (2) | 0 | 0 | 1 | 1 | 1 | 22 | 206 |
| Knapsack (3) | 1 | 0 | 1 | 0 | 0 | 13 | 58 |

Table 2.1: Coordinates in the decision space and in the objective space from 3 different knapsacks

## 2.3 Pareto Dominance

A point dominates an other point, *if it is better or equal in all components and better in at least one of the other component.*

The relation *is better* depends of the context. If we consider the objective space, and more specifically one of the objectives that have to be minimized, one of two points is better for this objective, if its value of this objective is stricly smaller.

If all the objectives have to be minimized, the Pareto dominance in the objective space can be written: $O^{(i)}$ dominates $O^{(j)}$ iff

$$\exists l \in [\![1,p]\!], \quad o_l^{(j)} < o_l^{(i)}$$

$$\text{and } \forall l \in [\![1,p]\!], \quad o_l^{(i)} \leq o_l^{(i)}$$

In the example introduced in Section 2.1, we have $2^5 = 32$ different possible knapsacks. It is easy to find a point which dominates other ones.

The introduced notation is made for objectives that have to be minimized. We can fit our knapsack problem into this model, by reversing the profit objective. Instead of maximizing the value, we minimize the objective : MAXVAL - value

MAXVAL is the maximal value that can be in archived (in this case $25 + 62 + 33 + 98 + 75 = 293$)

If we apply the previous definition to the knapsack (1), we see that other knapsacks are dominated. The coordinates are listed in the table 2.2 and displayed in Figure 2.3 .

| | Decision space | | | | | Objective space | |
|---|---|---|---|---|---|---|---|
| | *item in the knapsack?* | | | | | *weight* | *MAXVAL - value* |
| | $d_1$ | $d_2$ | $d_3$ | $d_4$ | $d_5$ | $o_1$ | $o_2$ |
| Knapsack (1) | 0 | 1 | 0 | 0 | 1 | 8 | 156 |
| dominated knapsack | 0 | 0 | 1 | 0 | 0 | 9 | 260 |
| | 0 | 0 | 1 | 0 | 1 | 15 | 185 |
| | 0 | 0 | 1 | 1 | 0 | 16 | 162 |
| | 0 | 1 | 1 | 0 | 0 | 11 | 198 |
| | 1 | 0 | 0 | 0 | 1 | 10 | 193 |
| | 1 | 0 | 0 | 1 | 0 | 11 | 170 |
| | 1 | 0 | 1 | 0 | 0 | 13 | 235 |
| | 1 | 0 | 1 | 0 | 1 | 19 | 160 |
| | 1 | 1 | 1 | 0 | 0 | 15 | 173 |

Table 2.2: Knapsacks dominated by the knapsacks (1)

## 2.4 Pareto Front

The Pareto front is the set of all the points that are not dominated by any of the other points. It is a quite important notion in the multiobjective optimization, because all the points in the Pareto front constitute the set of solutions to the given problem. Two solutions in the Pareto front are incomparable: if the first one is better in one objective, the second will be better in at least one other objective.

The decision maker has to select one of theses points, as an answer to his problem.

We can apply this definition to our knapsack problem instance, and extract the Pareto front (Figure 2.4).

We see that we found in this Pareto front the two trivial solutions: the empty knapsack (on the y-axis, the weight is 0) and the full knapsack (on the x-axis, MAXVAL - value = 0). All the other points of the Pareto front are possibilities that make a compromise between the two objectives.

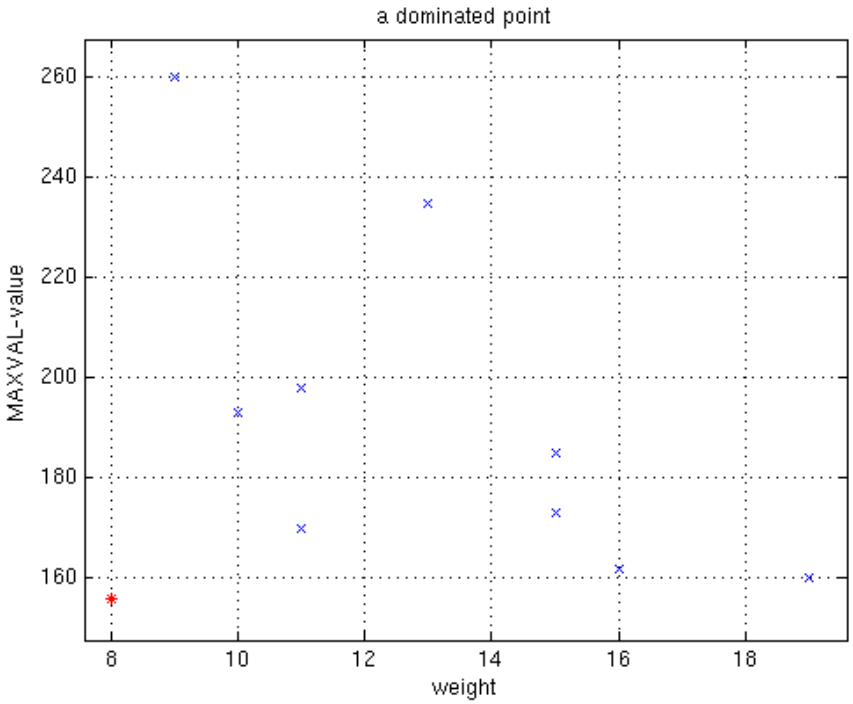


Figure 2.3: Knapsacks dominated by the knapsacks (1). $*$: the knapsacks (1); $\times$: the domniated knapsacks.

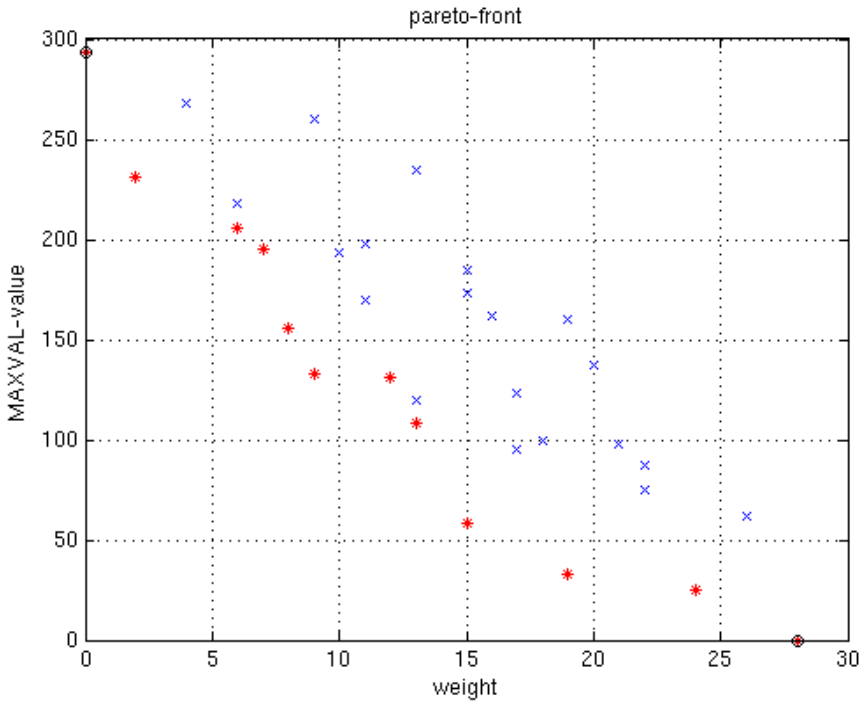


Figure 2.4: All possible solution of the Knapsack problem. $*$: Pareto Front; $o$: full and empty knapsack; $\times$: other knapsacks

# Chapter 3

# Dimension reduction

## 3.1 Introduction

### 3.1.1 Goals

Representing a two-dimensional space is something very common. We are used to representing it in a plane with a Cartesian coordinate system. Such a plane fits exactly a screen or a sheet of paper. Therefore, reducing a high dimensional space to a two-dimensional one might be good way to visualize it.

It is obvious that mapping a high dimensional space onto a two-dimensional one leads to an infinite among of possibilities. A lot of techniques have been developed, and are available. The goal is to select the one that is suitable in a multiobjective optimization's context.

The methods are very diversified and it is not easy to compare them. The common characteristic is their output: the set of points reduced in the two-dimensional space, which is the result of the dimension reduction techniques. Based on this result, and given the set of points in the high dimensional space, the mapping needs to be evaluated. 3 different metrics are proposed, in order to give a feedback of the goodness of a dimension reduction.

This enables to test different dimension reduction techniques, on several test sets and to choose the dimension reduction that is the most appropriate in our case.

### 3.1.2 Related work

Dimension reduction is a famous research topic. A lot of methods have been proposed. Amongst them we can distinguish:

Linear methods use linear combinations of variables to reduce the dimensions. A famous example of these methods is Principal Component Analysis. It is a special case of a more general method: projection pursuit.

Nonlinear methods work with combinations of nonlinear functions of variables. Algebraic Curve, Surface Fitting and Multidimensional scaling are examples of theses methods.

There are also methods based on topologically continuous maps. They use neuronal network trained on the data. Examples are Kohonen's self-organizing maps or Density Network.

According to the classification proposed in [2], we focus on dimension reduction applied to *visualization problems*. In this class of problems, the data do not have a very high dimension and are reduced to a space that can be plotted (dimension 2, 3 or at most 4). We restrain us to a mapping in two-dimensional spaces and we study only 2 of the methods that are often used in practice.

## 3.2 Approaches

### 3.2.1 Notation

As introduced in chapter 2, multiobjective optimization deals with 2 high dimensional spaces (the objective space and the decision space).

In this chapter we consider one $p$-dimensional space noted $\mathcal{A}$ (with $p \in \mathbb{N}$). This space is called original space.

$$\mathcal{A} \subset \mathbb{R}^p$$

In this space, the set of $n$ vectors is denoted:

$$\mathbf{A} = \{A^{(1)}, ..., A^{(n)}\}$$

The i-th point is denoted:

$$A^{(i)} = (a_1^{(i)}, a_2^{(i)}, ..., a_p^{(i)}) \in \mathbb{R}^p \quad 1 \leq i \leq n$$

A dimension reduction can been seen as a mapping function $t$ that goes from

the original space $\mathcal{A}$ to the reduced space $\mathcal{B}$.

$$t : \left| \begin{array}{ccc} \mathcal{A} & \rightarrow & \mathcal{B} \\ A^{(i)} & \mapsto & t(A^{(i)}) = B^{(i)} \quad 1 \leq i \leq n \end{array} \right.$$

In this case $\mathcal{B}$ has dimension 2:

$$\mathcal{B} \subset \mathbb{R}^2$$

The set of mapped vectors is noted:

$$\mathbf{B} = \{B^{(1)}, ..., B^{(n)}\}$$

The i-th point has following coordinates:

$$B^{(i)} = (b_1^{(i)}, b_2^{(i)}) \in \mathbb{R}^2 \quad 1 \leq i \leq n$$

### 3.2.2 Dimension reduction techniques

We focus on 2 dimension reduction techniques:

Principal Component Analysis (PCA) [5] is a linear method that combines the variables in order to construct a new orthonormal basis of the space. The first component is chosen in order to maximize the variance of the set of points projected on this direction. The next components proceed the same way with the additional constraint to stay orthogonal to the already selected components. The dimension is reduced as following: the dimensions with the smallest variance are discarded, i.e. the points are projected on the subspace spanned by the first 2 principal components. As a result, a good approximation of the original sample is obtained.

Mathematically the Principal Component Analysis can be seen as a spectral decomposition of the covariance matrix. For a reasonable number of dimensions, it is possible to compute such a decomposition, and in fact many implementations already exist For the test we used the Matlab function `princomp`, and obtained the coordinates in the reduced space by taking the 2 first colums of the resulting `SCORE` array.

According to [2], Principal Component Analysis (PCA) is in practice one of the wideliest used dimension reduction techniques, owing to its simplicity and the fact that efficient algorithms exist for its computations. However, there are limitations: the resulting reduced space is a linear subspace of the original one (which can be a problem for particular dataset), and in our case keeping only the 2 first components might be insufficient to give an accurate approximation of the dataset.

Kohonen's self-organizing maps (SOM) [6] is based on a neuronal network. According to [2] it is probably the best known technique, belonging to the Topologically Continuous Maps family. This technique's objective is to learn, in an unsupervised manner, a mapping from a grid of fixed dimensions (two in our case) onto the original space that embeds the data distribution. When the grid is trained: it maps as good as possible the original space and its members have coordinates in the reduced space (because of its fixed dimension). Each point $A^{(i)}$ in the original space has a nearest neighbor in the grid, whose coordinates in the reduced space are taken for the mapped point $B^{(i)}$.

Neuronal networks are very often used, and a lot of implementations already exist. For the tests, we took the Matlab implementation `newsom` from the Neural Network Toolbox. We chose a grid resolution of $12 \times 12$ and a training with 100 epochs.

Kohonen's self organizing maps have a major limitation: its heuristic nature. Training two times the same network on the same dataset gives two different results. This means there is no guaranty that it is possible to reproduce a good result obtained with such a dimension reduction technique.

## 3.3 Metrics

### 3.3.1 Properties

After a dimension reduction, each p-dimensional vector is represented by a two-dimensional vector. It is clear that such a mapping induces a lost of information. It is important to know what is the relevant information, in order to preserve it during the transformation. This enables to evaluate a transformation.

In a multiobjective optimization context, we focus on two main properties:

The first property concerns the distance between the points. If the points are considered pairwise, two points close in the original space should stay close to each other in the reduced space.

The second property deals with the dominance. A point which dominates others has a lot of signification. It helps to visualize the Pareto front. Preserving this dominance property in the reduced space is important.

In order to choose the best dimension reduction, we need to measure how these properties are preserved. That is the goal of the metrics proposed in the next Section.

### 3.3.2 Definition

Given a set of vectors in the original space $\mathbf{A}$ and a mapping function $t$, three metrics $D_1(\mathbf{A}, t)$, $D_2(\mathbf{A}, t)$ and $D_3(\mathbf{A}, t)$ are developed. They quantify how good the properties identified in 3.3.1 are preserved.

#### 3.3.2.1 Metric $D_1$

The metric $D_1$ measures how the distances are preserved.

**Step 1:** normalization of the space. Each dimension is scaled to the $[0, 1]$ interval. If the all values are the same in a dimension, then it is normalized to 0.

$$\overline{a}_i^{(j)} = \begin{cases} 0 & \text{if} \quad \forall l, m \in [\![1, p]\!], \quad a_i^{(l)} = a_i^{(m)} \\ \frac{a_i^{(j)} - \min_{1 \leq l \leq n}(a_i^{(l)})}{\max_{1 \leq m \leq n}(a_i^{(m)}) - \min_{1 \leq l \leq n}(a_i^{(l)})} & \text{else} \end{cases}$$

The normalized points are denoted:

$$\overline{A}^{(i)}, \quad 1 \leq i \leq n$$

**Step 2:** In the original and the reduced space the Euclidian distances are pairewise computed. The distances are weighted with the size of their space. The quotient between the weighted distance in the original space and in the reduced space is computed. The quotient should stay as near as possible to 1. If it is bigger than 1, the inverse is taken.

$$\text{for } 1 \leq i < j \leq k$$

$$s_{i,j} := \begin{cases} \frac{\frac{\|\overline{a}^{(i)} - \overline{a}^{(j)}\|}{\sqrt{n}}}{\frac{\|\overline{b}^{(i)} - \overline{b}^{(j)}\|}{\sqrt{2}}} & \text{if} \quad \frac{\|\overline{a}^{(i)} - \overline{a}^{(j)}\|}{\sqrt{n}} < \frac{\|\overline{b}^{(i)} - \overline{b}^{(j)}\|}{\sqrt{2}} \\ \frac{\frac{\|\overline{b}^{(i)} - \overline{b}^{(j)}\|}{\sqrt{2}}}{\frac{\|\overline{a}^{(i)} - \overline{a}^{(j)}\|}{\sqrt{n}}} & \text{if} \quad \frac{\|\overline{a}^{(i)} - \overline{a}^{(j)}\|}{\sqrt{n}} > \frac{\|\overline{b}^{(i)} - \overline{b}^{(j)}\|}{\sqrt{2}} \\ 1 & \text{else} \end{cases}$$

**Step 3:** The quotients are sumed up and normalized.

$$D_1(\mathbf{A}, t) := \frac{2}{n(n-1)} \sum_{i=1}^{n} \sum_{j=i+1}^{n} s_{i,j}$$

The metric is good when it equals 1.

#### 3.3.2.2 Metric $D_2$

The metric $D_2$ focuses on the dominance property.

**Step 1:** For each point in the original space the set of points that it dominates is searched.

$$\mathrm{Dom}_A(i) = \{j \in [\![1, n]\!],\ A^{(i)} \text{ dominates } A^{(j)}\}$$

The complement set:

$$\mathrm{Dom}_A^*(i) = \{j \in [\![1, n]\!],\ A^{(i)} \notin \mathrm{Dom}_A(i)\}$$

**Step 2:** For each point in the reduced space the set of points that it dominates is searched.

$$\mathrm{Dom}_B(i) = \{j \in [\![1, n]\!],\ B^{(i)} \text{ dominates } B^{(j)} = t(A^{(j)})\}$$

The complement set:

$$\mathrm{Dom}_B^*(i) = \{j \in [\![1, n]\!],\ A^{(i)} \notin \mathrm{Dom}_B(i)\}$$

**Step 3:** A point that is dominated in the original space has to stay dominated in the reduced space. The number of changes is sumed up and normalized.

$$D(\mathbf{A}, t) = \frac{1}{n(n-1)} \sum_{i=1}^{n} \left(|\mathrm{Dom}_A(i) \cap \mathrm{Dom}_B^*(i)| + |\mathrm{Dom}_B(i) \cap \mathrm{Dom}_A^*(i)|\right)$$

The metric is good when it equals 0.

#### 3.3.2.3 Metric $D_3$

The metric $D_3$ compromises the 2 properties.

**Step 1:** Normalization of the two spaces (same step as for $D_1$ in 3.3.2.1).

**Step 2:** $s_A(i, j)$ measures on each component how much $A^{(i)}$ has to be moved in order to dominate $A^{(j)}$ in the original space.

$$s_A(i, j) = \frac{1}{p} \sum_{l=1}^{n} \left(\max\left(0, \overline{a}_l^{(i)} - \overline{a}_l^{(j)}\right)\right)$$

**Step 3:** $s_B(i, j)$ measures on each component how much $B^{(i)}$ has to be moved in order to dominate $B^{(j)}$ in the reduced space.

$$s_B(i, j) = \frac{1}{2}\left(\max\left(0, \overline{b}_1^{(i)} - \overline{b}_1^{(j)}\right)\right) + \frac{1}{2}\left(\max\left(0, \overline{b}_2^{(i)} - \overline{b}_2^{(j)}\right)\right)$$

**Step 4:** $t(i,j)$ is computed. $t(i,j)$ is the quotient of the previously computed distances, with some special cases.
if:

- $0 < s_A(i,j) < s_B(i,j)$
$$t(i,j) = \frac{s_A(i,j)}{s_B(i,j)}$$
- $0 < s_B(i,j) < s_A(i,j)$
$$t(i,j) = \frac{s_B(i,j)}{s_A(i,j)}$$
- $s_A(i,j) = 0$ and $s_B(i,j) > 0$
$$t(i,j) = 0$$
- $s_B(i,j) = 0$ and $s_A(i,j) > 0$
$$t(i,j) = 0$$
- $s_A(i,j) = 0$ and $s_B(i,j) = 0$
$$t(i,j) = 1$$

**Step 5:** Sum the computed quotient and normalize.

$$D(\mathbf{A},t) = \frac{1}{n(n-1)} \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} t(i,j)$$

The metrics is good when it equals 1.

### 3.3.3 Example

A simple example helps to understand the metrics introduced in 3.3.2. We consider 5 points in the original space, whose size is $p = 3$. Figure 3.1 represents the points with parallel coordinates and the table 3.1 gives the coordinates of each point in the space.

We propose 3 different matching ($t_1$, $t_2$, $t_3$) for the proposed set of point. The 3 resulting sets of reduced points are presented in Figures 3.2, 3.3 and 3.4. The table 3.2 presents the scores with the 3 metrics of the different transformations for the proposed set of points.

The transformations are nonlinear. They are built on the proposed set of points in the original space: for each point the position of the matching point in the reduced space is chosen. Each transformation can be seen as a function with 10 degrees of freedom: the coordinates (2 for each point) of the 5 points.

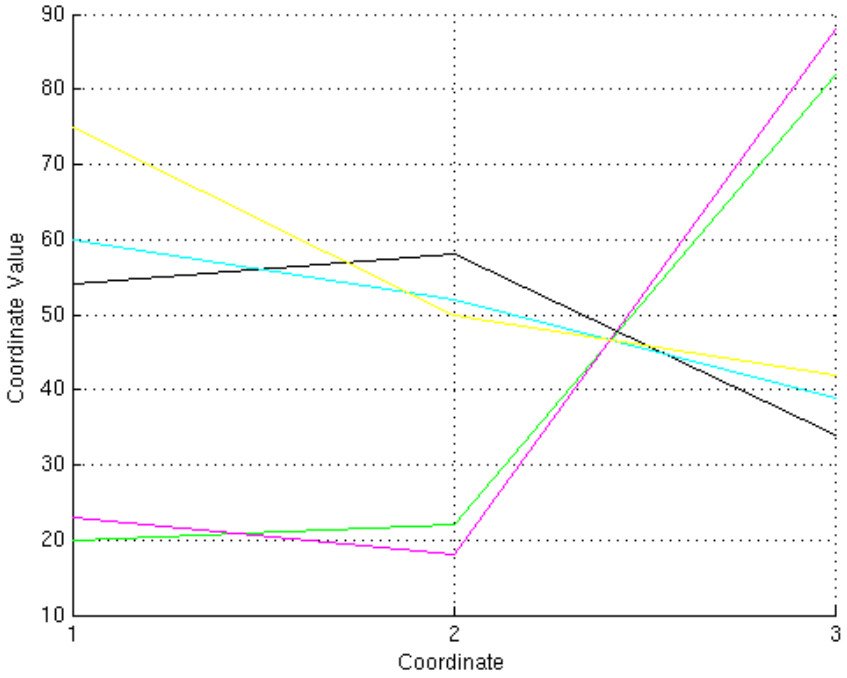


Figure 3.1: Five points in the original space represented with Parallel coordinates

| **A** | Original space | | |
|---|---|---|---|
| | $a_1$ | $a_2$ | $a_3$ |
| $A^{(1)}$ | 20 | 22 | 82 |
| $A^{(2)}$ | 23 | 18 | 88 |
| $A^{(3)}$ | 54 | 58 | 34 |
| $A^{(4)}$ | 60 | 52 | 39 |
| $A^{(5)}$ | 75 | 50 | 42 |

Table 3.1: Coordinates of the five points in the original space

Each one of the proposed matchings tries to optimize one of the metrics introduced in Section 3.3.1. The optimization problem was not studied in details: we applied a simple Localized Random Search with a good initialization step (based on intuition). It probably exists better matching for $t_1$ and $t_3$, but the proposed transformations are good enough to show the purpose.

The distances between the points are very good preserved in the first matching (Figure 3.2). In the original space, the yellow, blue and black points are close to each others and the green and purpule points are close to each other. These 2

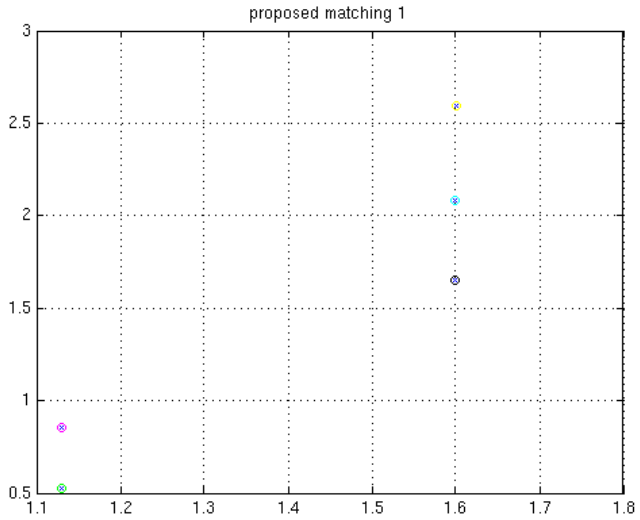


Figure 3.2: First proposed matching ($t_1$)

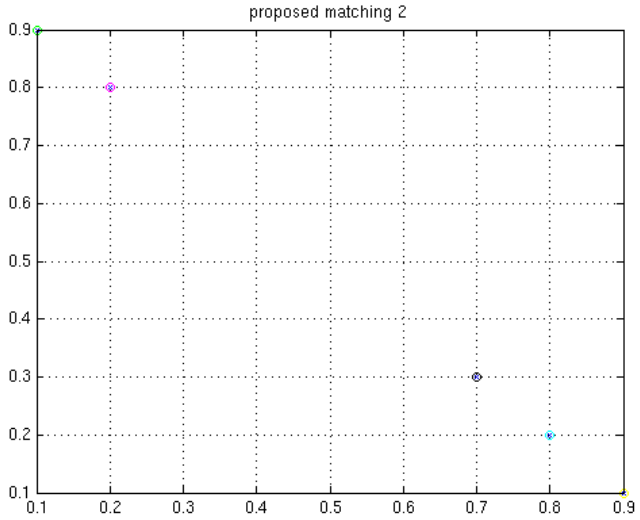


Figure 3.3: Second proposed matching ($t_2$)

| | | |
|---|---|---|
| $D_1(\mathbf{A}, t_1) = 0.901213$ | $D_2(\mathbf{A}, t_1) = 0.400000$ | $D_3(\mathbf{A}, t_1) = 0.668244$ |
| $D_1(\mathbf{A}, t_2) = 0.799368$ | $D_2(\mathbf{A}, t_2) = 0.000000$ | $D_3(\mathbf{A}, t_2) = 0.400000$ |
| $D_1(\mathbf{A}, t_3) = 0.428210$ | $D_2(\mathbf{A}, t_3) = 0.100000$ | $D_3(\mathbf{A}, t_3) = 0.990755$ |

Table 3.2: Measure of the proposed matching with the metrics

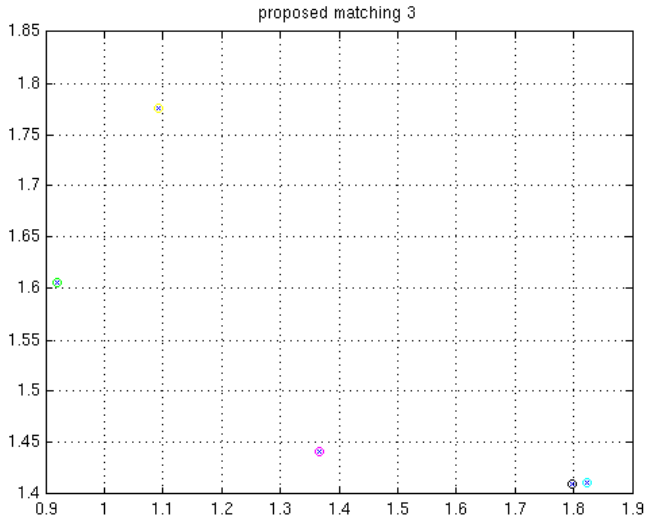


Figure 3.4: Third proposed matching ($t_3$)

groups are also present in the reduced space. The 5 points constitute a Pareto front: no point dominates an other one. This is also the case in the second reduced space (Figure 3.3). The distance $D_2(\mathbf{A}, t_2)$ is optimal. The third matching is a compromise between the 2 properties (Figure 3.4).

This example constitutes a simple verification of the proposed metrics' correctness. We can use them to evaluate 2 different dimension reduction procedures and choose the best one.

## 3.4 Tests

### 3.4.1 Test set

For the tests we dispose of some instances of the extended knapsack problem introduced in the chapter 2. They were produced with the PISA platform [1]. To create different instances, two parameters were modified:

- The number of objectives is 2, 3 or 4 (This modifies the size of the objective space).
- The number of design parameters is 50, 100, 150 or 200 (This modifies the size of the decision space).

For each of the 12 problems, the best solutions were searched with a multiobjective optimization algorithm. The result of each optimizer is a Pareto front of the given problem. The coordinates of each point are given in the objective space and in the decision space. The two spaces are independent and are considered as two separated high dimensional spaces.

### 3.4.2 Results

For each space of each test set we applied the two dimension reduction techniques introduced in 3.2.2 (Principal Component Analysis and Kohonen's self-organizing maps). We measured the quality of the obtained dimension reduction with the metrics introduced in 3.3.2. For the decision space, only the metric $D_1$ has a signification, because there is no dominance relation in the decision space.

For each metrics and each space the results are grouped and presented with box plots in Figures 3.5, 3.6, 3.7 and 3.8.

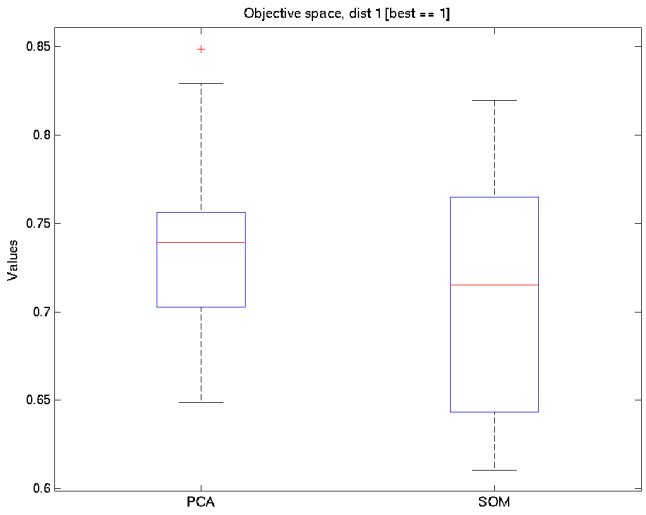


Figure 3.5: Results in the objective space, measured with the metric $D_1$ for PCA and SOM over the 12 test sets.

For the objective space, Figure 3.5 shows that distances are circa as well preserved in the objective space with the two transformations. The median is quite the same, but the quartiles are narrower for the PCA, which is better. Figure 3.6 shows that the dominance relation is much better preserved with the PCA. For the $D_3$ metric, we see Figure 3.7 that both transformations have very bad and dispersed results. It is not possible to distinguish the transformation. For the objective space

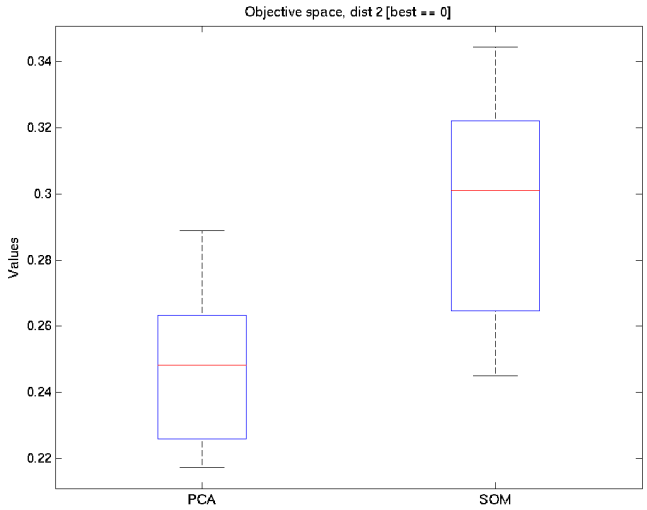


Figure 3.6: Results in the objective space, measured with the metric $D_2$ for PCA and SOM over the 12 test sets.

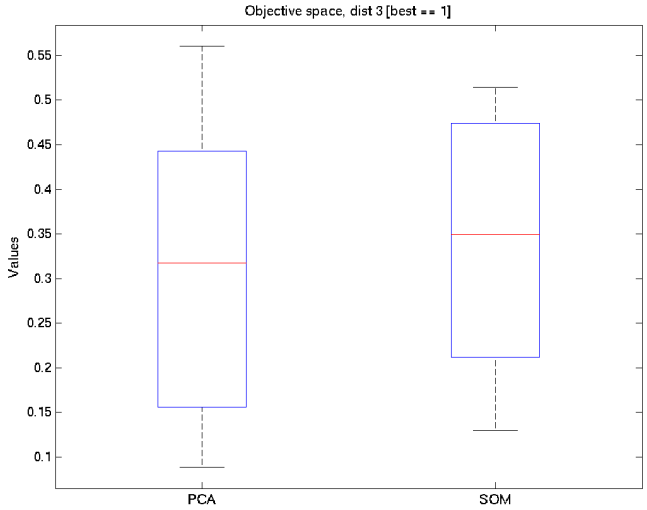


Figure 3.7: Results in the objective space, measured with the metric $D_3$ for PCA and SOM over the 12 test sets.

PCA seems to be a good transformation.

In the decision space the reduction is much stronger (in the worst case we

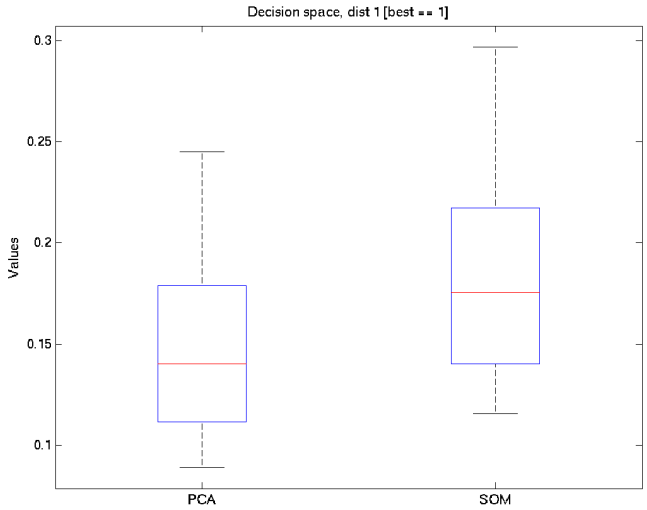


Figure 3.8: Results in the decision space, measured with the metric $D_1$ for PCA and SOM over the 12 test sets.

reduced 200 dimensions onto a two-dimensional space). The results Figure 3.8 are bad for the 2 transformations. An explanation is maybe that the original space is binary (discrete), but considered by the dimension reduction techniques as a continuous space. The SOM transformation seems to be a little bit better, but it is not outstanding.

Considering theses results, PCA seems to be as good as SOM in most of the cases and sometimes better. This dimension reduction seems to be suitable in a multiobjective optimization context.

## 3.5 Conclusion and future work

In this chapter we gave 3 different metrics in order to evaluate how good a dimension reduction is in a context of high dimensional spaces coming from multiobjective optimization. Each metric measures how good a matching is for a given dataset, with respect to some interesting properties. This enables to choose between different matchings.

Two very different, but widely used dimension techniques were compared: Principal Components Analysis is a linear reduction dimension techniques and Kohonen's self-organizing maps is based on a neuronal network. Principal Components Analysis got better results with the metrics, and is chosen to be implemented in

the visualization tool.

The test sets produced with the knapsack problem may not be representative of all multiobjective optimization datasets. Other data (bigger objective space, continuous decision space) have to be tried.

There is a lot of available dimension reduction techniques. We provide a way to compare them given a test set. They have to be tried in order to see if something better already exists.

The proposed metrics enable to quantify how good interesting properties are preserved. This might enable the development of new dimension reduction techniques as an optimizer: the parameters are the position of the points in the reduced space and the objective is to minimize or maximize the score of the proposed metrics.

# Chapter 4

# Clustering

## 4.1 Introduction

### 4.1.1 Goals

When we are dealing with a lot of points, grouping them in order to get a better overview is a natural idea. If the groups are well built, this reduces the comparison's problem: only the groups of points need to be considered in a first step. In a second approach, it is possible to focus only on a group of points.

We want to apply this strategy to solve the problem of the large number of points obtained as solutions of a multioptimisation problem (the Pareto front). The points lie simultaneously into two spaces (the objective space and the decision space). Building similarity groups in the objective space is interesting, because members of a group will have close scores in the different objectives (that are directly linked with the optimization problem). However, building clusters in the decision space is also interesting. This can help the designer to understand the importance or the effect of a parameter. In the case of the knapsack problem, the decision variable expresses the presence of an item in the knapsack. Knowing that in the amount of optimal solutions a group of knapsacks contains the same items is a valuable information. We are interested in groups in the objective space and in the decision space.

Similarity groups are called clusters. This chapter will study how a clustering procedure can be applied to the solutions of an optimization problem. We focus on algorithms that produce a partition of the set of points. We will study how the clustering procedure can combine the two spaces.

### 4.1.2 Related work

Clustering is a well known problem. A lot of algorithms have been proposed to construct meaningfull groups of points.

A lot of them only work in a single high dimensional space. $k$-means [10] is based on nearest distances. Hierarchical clustering [4] is an iterative procedure that always groups the two closest objects. But these procedures need to be adapted to work in two dimensions.

Ping pong [8] clustering procedure works with 2 spaces, but only sorts out one group of points. Considering a partition with only 2 subsets (elected and not-elected points) is not very satisfying.

In the rest of the chapter we give first some definition of a cluster. After that we propose an extension of the $k$-means clustering procedure in order to be able to work in 2 spaces. In the last Section the procedure is tested on instances of the knapsack problem.

## 4.2 Cluster

### 4.2.1 Notation

We consider a set of points, noted:

$$\mathbf{P} = \{P^{(1)}, P^{(2)}, ...P^{(n)}\}$$

As introduced in Section 2.2, each point has coordinates in the objective space and in the decision space.

In this chapter the clustering procedure produces $k$ groups of points, called clusters. Each cluster contains some of the points. Mathematically they are subset of the set of points:

$$K_1, K_2, K_3, ...K_k$$
$$K_i \subset \mathbf{P}, \quad 1 \leq i \leq k$$

All clusters build a partition of the set of points:

$$\bigcup K_i = \mathcal{P}$$
$$K_i \cap K_j = \emptyset \quad \text{if} \quad i \neq j$$

The clustering procedure we study in this chapter can be seen as an algorithm that provides a partition of a given set of points.

### 4.2.2 Evaluation of a partition

With the given definition of a clustering procedure, the results can be very different. It is important to be able to evaluate a partition.

We define such a metric for each space (the objective space and the decision space).

**Step 1**: each space is normalized (see step 1 of $D_1$ in 3.3.2.1)

**Step 2**: the key idea is that for each cluster the coordinates of the points should stay as near as possible. Therefore for each dimension the interval containing all the values of the cluster should be as small as possible. The size of the interval equals the difference between the maximum and the minimum. The metric is the normalized sum. For the decision space:

$$d_{\text{obj}} = \frac{1}{k}\sum_{i=1}^{k}\frac{1}{p}\sum_{j=1}^{p}\left(\max_{l:P_l\in K_i}(\overline{o}_j^{(l)}) - \min_{l:P_l\in K_i}(\overline{o}_j^{(l)})\right)$$

For the decision space:

$$d_{\text{dec}} = \frac{1}{k}\sum_{i=1}^{k}\frac{1}{q}\sum_{j=1}^{q}\left(\max_{l:P_l\in K_i}(\overline{d}_j^{(l)}) - \min_{l:P_l\in K_i}(\overline{d}_j^{(l)})\right)$$

The two metrics are exactly computed the same way in their respective space. With the proposed normalization factors, it holds : [0, 1]. The partition is better if the metric equals 0.

In order to be able to give one evaluation of the partition for the two spaces, we combine $d_{\text{obj}}$ and $d_{\text{dec}}$ with a parameter $\beta \in [0, 1]$. $\beta$ expresses the importance of the objective space over the decision space.

$$d = \beta d_{\text{obj}} + (1-\beta)\, d_{\text{dec}}$$

- $\beta = 1$: evaluation of the objective space only.
- $\beta = 0$: evaluation of the decision space only.

## 4.3 Clustering procedure

### 4.3.1 $k$-means Algorithm

Clustering is one of the main applications of the $k$-means algorithm [10]. It delivers qualitative and quantitative reasonably good similarity groups, which helps to

understand large amount of high dimensional data.

It can be seen as a very simple iterative procedure represented in the flowcharts in Figure 4.1.

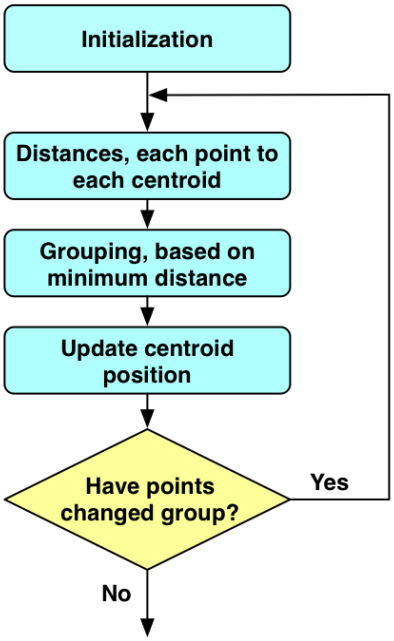


Figure 4.1: Flowchart representation of the $k$-means algorithm

During the initialization step, $k$ centroids are randomly chosen. A centroid is a vector living in the same space as the points (normalized).

$$\overline{C}^{(1)}, \overline{C}^{(2)}, \overline{C}^{(3)}, ...\overline{C}^{(k)}$$

$$\overline{C}^{(i)} \in [0,1]^p$$

The iteration process is composed of 3 simple steps.

During the first step, the Euclidian distance between each point and each centroid is computed.

$$d(\overline{A}^{(i)}, \overline{C}^{(j)}) = \frac{1}{p} \sum_{l=1}^{p} (\overline{a}_l^{(i)} - \overline{c}_l^{(j)})^2$$

The second step groups the points together. Each point is associated with the centroid to which it is the nearest.

$$K_i = \{A^{(j)} \in \mathbf{A} | \overline{C}^{(i)} \text{ is the nearest centroid of } \overline{A}^{(j)}\}$$

In the third step, the centroids are moved: each centroid is the mean of the points in the cluster it represents.

$$\overline{c}_l^{(i)} = \frac{1}{|K_i|} \sum_{A^{(j)} \in K_i} \overline{a}_l^{(j)}, \quad 1 \leq l \leq p$$

The iteration continues until all the points stay in the same cluster as the one where they were in the previous round.

### 4.3.2 Extension to two spaces

The presented algorithm is made for a single space. The points we are dealing with have coordinates in the objective space and in the decision space. The objective is to extend this algorithm to be able to work simultaneously in two spaces.

The approach is very simple: the centroid gets the same structure as the points (coordinates in 2 the spaces: $\overline{C}_{\text{obj}}^{(i)}$ and $\overline{C}_{\text{dec}}^{(i)}$ ). The algorithm can work in parallel with the two high dimensional spaces (objective space and decision space), exactly like in the simple case.

The single operation that needs to be redefined is the computation of the distances. The distance has now to be computed between each point consisting of an objective vector and a decision vector and each centroid consisting also of two vectors. We introduce a parameter $\alpha \in [0, 1]$ and compute a weighted sum:

$$d(\overline{P}^{(i)}, \overline{C}^{(j)}) = \alpha d(\overline{O}^{(i)}, \overline{C}_{\text{obj}}^{(j)}) + (1 - \alpha)\, d(\overline{D}^{(i)}, \overline{C}_{\text{dec}}^{(j)})$$

The parameter $\alpha$ is very similar to the parameter $\beta$ introduced in the the Section 4.2.2. To execute the clustering procedure in the two spaces without giving more importance to one space, we take $\alpha = \frac{1}{2}$

Notice that $\alpha$ and $\beta$ can have different values: clusters can be built in the two spaces and only evaluated in one of the two.

Figure 4.2 shows the evolution of the evaluation of the clusters functions of the iterations with $\alpha = \frac{1}{2}$. We see that each iteration optimizes the evaluation of the current clusters.

### 4.3.3 Improvement

Already in its original version, it is known that the $k$-means algorithm does not always converge to an optimal partition [10]. Because of its construction, it can get

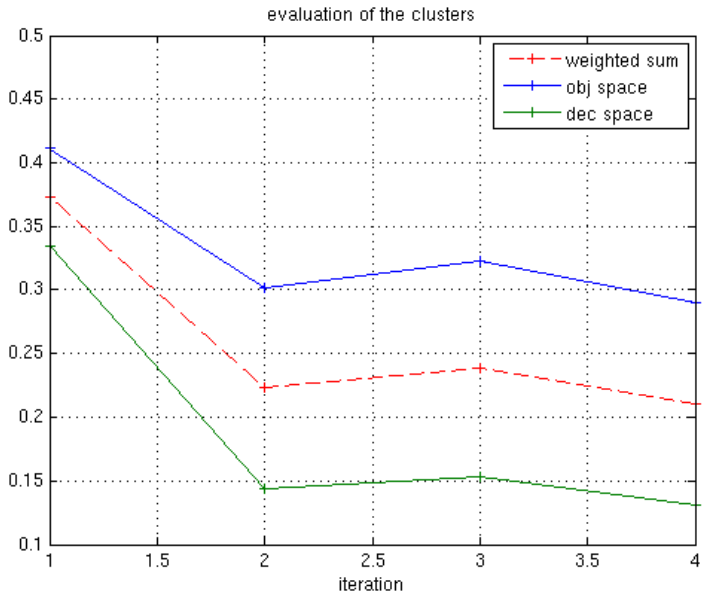


Figure 4.2: Evaluation during the $k$-means clustering procedure (with $\alpha = 0,5$). In blue evaluation with $\beta = 1$, in red $\beta = 0,5$ and in green $\beta = 0$

stuck in a local minimum. The only randomized part is the initialization step. After that in a few deterministic rounds it converges to a local minimum.

To avoid this major drawback, the following strategy is proposed: the algorithm is executed $N$ times, and the best solution is kept. This makes the algorithm robuster.

## 4.4 Tests

### 4.4.1 Influence of parameter $\alpha$

We want to verify that the proposed algorithm with the parameter alpha, is better than a simple optimization of the objective space. The evaluation of the partition is made in the objective space only ($\beta = 1$) and in the decision space only ($\beta = 0$). The parameter $\alpha$ balances the construction of the cluster between the objective space and the decision space. 3 values of $\alpha$ are tested: $\alpha = 0$ , $\alpha = 0,5$ and $\alpha = 1$.

We expect that with $\alpha = 1$, the proposed partition is better in the objective space ($d_{\text{obj}}$ smaller). On the contrary with $\alpha = 0$ the proposed partition is better in the decision space ($d_{\text{dec}}$ smaller).

We used the same test set as in Section 3.4.1. The $k$-means clustering procedure needs an input parameter $k$. We make tests with the value $k = 3$, 5 and 10. For each instance of the knapsack problem the clustering procedure is executed 20 times. The results are presented in boxplot in the next Section.

### 4.4.2 Results

We get 36 different plots and it is hard to sort them. We propose to split the results in 3 classes.

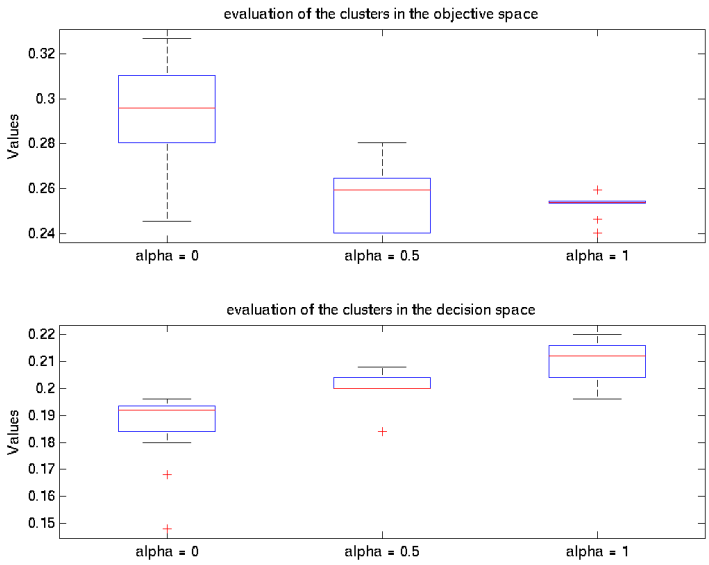


Figure 4.3: Evaluation of the clusters (5 clusters , 4 objectives, 50 decision parameters). $\alpha = 0$: only decision space, $\alpha = 0,5$: two spaces, $\alpha = 1$: only objective space. Evaluation in the objective space ($\beta = 1$) and in the decision space ($\beta = 0$).

The first class of result can be qualified as *very good*. For the objective space ($\beta = 1$, top graphs), if the objective space is taken into consideration ($\alpha \neq 0$), we get better results (the metric is smaller) than if the space is not. The results are the best for $\alpha = 1$ (only the objective spaces are taken into consideration). For the decision space ($\beta = 0$, bottom graph), the results are symmetric: the results are the worse for $\alpha = 1$, and the best for $\alpha = 0$. Figure 4.3 shows an example. To sort results in this class, we had a look at the median.

- In the objective space: $M_{(\alpha=0)} \geq M_{(\alpha=0,5)} \geq M_{(\alpha=1)}$

- In the decision space: $M_{(\alpha=0)} \leq M_{(\alpha=0,5)} \leq M_{(\alpha=1)}$

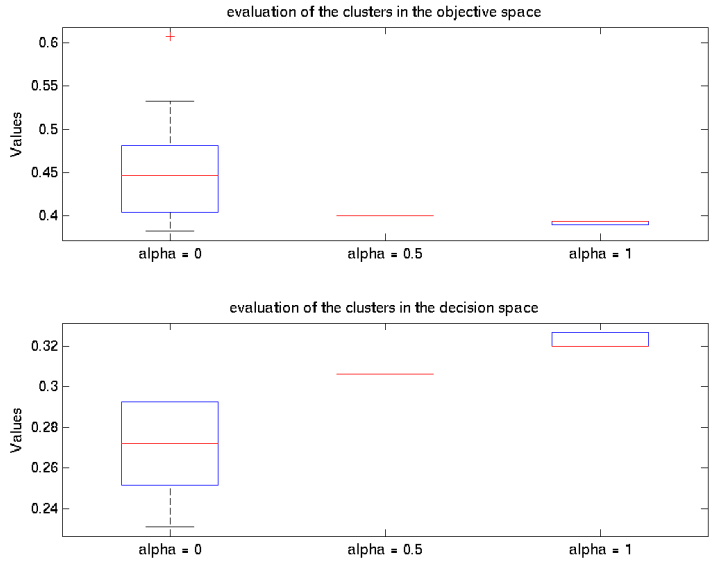


Figure 4.4: Evaluation of the clusters (3 clusters , 3 objectives, 50 decision parameters)

For some of the results , we sometimes get always the same ones. In the boxplot graph, this is expressed with a single red line. An explanation could be that the $k$-means algorithm converges always to the same partition (maybe the global optimum). Figure 4.4 shows an example of such a result which is also classified in the first class.

The second class of results concerns the *almost good* ones. In comparison with the results of the first class, we allow that some of the medians are not as well sorted. Sometimes the distribution of the points is average good, but it may have been got with this parameters results that did not behave as expected. One example is given in Figure 4.5.

The third class deals with *bad* results. We have distributions that are contrary to the expected ones. These cases are very complicated to analyze. An explanation could be that the algorithm did not manage to build stable partition. An other possibility might be that the objective space and decision space are very dependent: building clusters in one space makes optimization in the two spaces. Figure 4.6 shows an example.

The Table 4.1 gives the classification of each result.

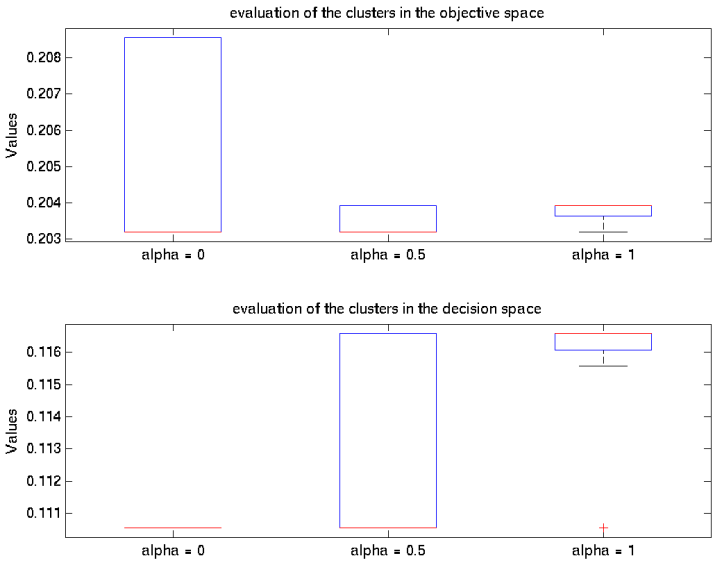


Figure 4.5: Evaluation of the clusters (5 clusters , 3 objectives, 200 decision parameters)

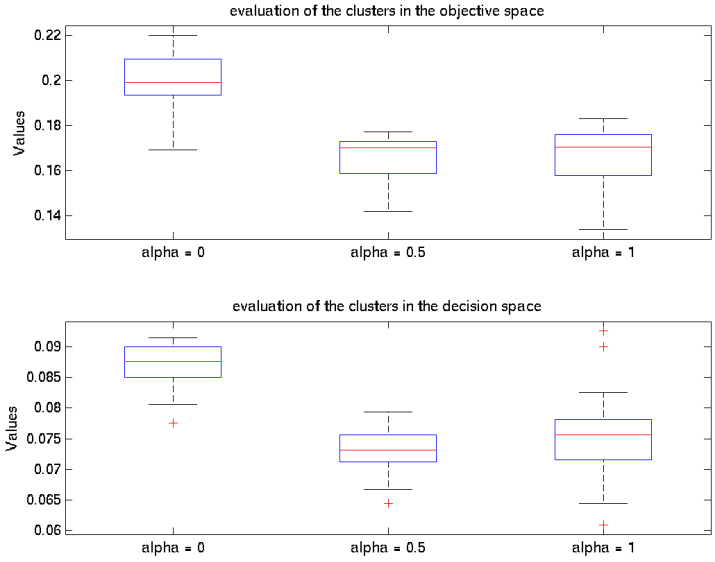


Figure 4.6: 10 clusters , 4 objectives, 200 decision parameters

| | $p = 2$ | $p = 3$ | $p = 4$ |
|---|---|---|---|
| $q = 50$ | $k = 3$: very good<br>$k = 5$: good<br>$k = 10$: very good | $k = 3$: very good<br>$k = 5$: good<br>$k = 10$: very good | $k = 3$: bad<br>$k = 5$: very good<br>$k = 10$: good |
| $q = 100$ | $k = 3$: very good<br>$k = 5$: bad<br>$k = 10$: good | $k = 3$: good<br>$k = 5$: good<br>$k = 10$: good | $k = 3$: bad<br>$k = 5$: good<br>$k = 10$: good |
| $q = 150$ | $k = 3$: very good<br>$k = 5$: very good<br>$k = 10$: good | $k = 3$: very good<br>$k = 5$: bad<br>$k = 10$: good | $k = 3$: bad<br>$k = 5$: good<br>$k = 10$: good |
| $q = 200$ | $k = 3$: bad<br>$k = 5$: good<br>$k = 10$: very good | $k = 3$: bad<br>$k = 5$: good<br>$k = 10$: good | $k = 3$: good<br>$k = 5$: bad<br>$k = 10$: bad |

Table 4.1: Classification for each result. $p$: size of the objective space; $q$: size of the decision space, $k$: number of clusters

## 4.5 Conclusion and future work

In this chapter we have proposed a method to be able to evaluate how good a partition resulting of a clustering procedure is. We have extended a simple clustering procedure in order to work in the objective space and in the decision space simultaneously. We have tested the proposed algorithm on some instances of the knapsack problem with different configurations. As a result we show that in most of the cases the parameters $\alpha$ introduced to balance the spaces has a real effect on how the clusters are built. Setting $\alpha = \frac{1}{2}$ in order to build cluster in the 2 spaces simultaneously, modifies the construction of the clusters.

A lot of other clustering procedures are available. They should also be tested. The proposed metric provides a good way to evaluate the results. If a clustering procedure only works in one space, the same method can be applied in order to extend it to the objective space and the decision space. Other extensions can also be developed, like optimizing alternately in one space and in the other space.

# Chapter 5

# Visualisation Tool

## 5.1 Introduction

### 5.1.1 Goals

Visualizing solutions provided by a multiobjective optimization problem is crucial for the decision-making. A specific tool is proposed to display that kind of points. We are interested in the coordinates of the points in the objective space and in the decision space. This is not an easy task because the two spaces are high dimensional.

The work done in the two previous chapters highlights some techniques to handle two problems appearing with this type of set of points. The implemented tool has to provide some dimension reduction techniques and some clustering procedures.

The final product is a simple Java application. It should constitute a basis that can be reused for future developments.

### 5.1.2 Related work

Providing the decision maker a good representation of the points has been studied by a lot of authors. Korhonen and Wallenius [7] give a good overview of the different possibilities in the general case and in a multiple objective problem context. For the representation of the whole set, a parallel coordinate plot (called score profile) is a standard representation. If the set of points is small enough it can be represented with bar graphs. They also propose an approach where each solution is symbolized with a picture in order to make pairwise comparisons. Lotov and Miettinen [9] propose sophisticate methods to represent the objective space.

The 2 papers focus on the objective space. [9] mentions that the decision space can be so big (many thousand dimensions) that it would prevent it from being visualized. However in our case the decision space is not so big and can be represented.

## 5.2 Requirements

### 5.2.1 PISA Compatibility

PISA is a platform constituting a text-based interface for search algorithms. It enables to plug-in any optimization problem with any optimization algorithm. For the visualization tool, assuring a PISA compatibility means being able to open the set of points in a specific file format.

### 5.2.2 Representation of the points

We are dealing with two high dimensional spaces: the objective space and the decision space. The tool has to provide a lossless and a reduced representation of each space. The dimension reduction techniques have to map the high dimensional points in a two dimensional space.

The tool has also to consider the clustering procedures. They produce a partition of the given set of points. Such a partition has to be represented in the tool.

The algorithms studied in the two previous chapters have to be implemented, in order to offer basic functionalities. For the lossless representation, the parallel coordinates plot is used. The implemented dimension reduction procedure is principal component analysis. And the tool has to provide a $k$-means clustering procedure.

### 5.2.3 Extensible architecture

We have seen in the previous chapters, that a lot of dimensions reduction techniques and clustering procedure were available. Therefore it is important that the chosen architecture can be easily extended.

The user might want to use a different reduction technique or change the clustering procedure. For the programmer it has to be easy to plug-in these new possibilities.

## 5.3 Design choices

### 5.3.1 Organisation of the main windows

The main window (screeshot in Figure 5.1) contains 4 graphs: 2 for the objective space and 2 for the decision space. Each space is represented with a lossless representation (e.g. parallel plots) and a reduced representation (e.g. principal component analysis). Spatially the graphs are organized in a $2 \times 2$ grid. The first row is for the objective space, and the second row is for the decision space. The first column is for the lossless representation and the second column for the reduced space.

The partition obtained after a clustering procedure is represented with color. Of course the color of a point is the same in the four representations. The main window can display the whole set of points or only a subset constituted with the points in one or more clusters. The coloration of the point can be disabled

### 5.3.2 Structure

In order to be able to manipulate the data, the following structure is used.

**Coordinate:** the coordinate object is used to store a $n$-dimensional coordinates' vector.

**Point:** the point object contains two coordinate objects (one for each space). If the input file also provides a unique id for each point, this information is also stored.

**Set:** a set object is a collection of point objects. A set is used to store the whole population.

**Partition:** a partition object is a collection of set objects with the following constraint: a point can't be simultaneously member of the two subsets of the partition.

### 5.3.3 Procedures

In the chosen architecture the 3 main functionalities are seen as procedures:

**Clustering:** A clustering procedure takes as input the set containing all the points and produces a specific partition of this set.

**$n$-dimensional plot:** A $n$-dimensional plotting procedure takes a partition of the points as input and displays it in a `javax.swing.JPanel`.

**2-dimensional plot:** A 2-dimensional plotting procedure also takes a partition of the points as input and displays it in a `javax.swing.JPanel`, but it contains additionally a dimension reduction method.

A template is proposed for each type of procedure. Any new procedure has to respect this pattern in order to be plugged in the application. See the *Developer Guide* B for more information.

### 5.3.4 User interaction

The user interaction is very standard: to do an action, the user manipulates some modal dialogs where he can define the parameters. The modal dialogs that enable to chose a procedure (clustering or plotting) are similar: On the top: a combo-box enables to choose between the available procedures. This guaranties that a new procedure will easily be plugged in: a new line appears in the menu.

Some procedures need parameters in order to work. This means that a part of the graphical user interface depends on the selected procedure. Each procedure can define how the modal dialog looks like. Example and tutorials are given in the *Developer Guide* B.

### 5.3.5 Library

Some of the features are provided by existing libraries:

**JFreeChart:** JFreeChart[1] is a library for the creation of charts under LGPL license. It is used in the different plotting procedures in order to produce the graphs. JFreeChart is a very popular project in the Java community, used in a lot of applications. It supports a wide range of chart type and provides a good API. It is compatible with the Swing[2] components.

**JAMA:** JAMA[3] is a basic linear algebra package for Java. It is used for the Principal Component Analysis dimension reduction procedure.

[1] http://www.jfree.org/jfreechart/

[2] Swing is part of Sun Microsystems' Java Foundation Classes. It provides a sophisticate graphical user interface (GUI) for Java programs

[3] http://math.nist.gov/javanumerics/jama/

## 5.4 Screenshot

Figure 5.1 is a screenshot of the main window. More information about the user's interface can be found in the *User Guide* A given in the appendix.


Figure 5.1: Screenshot of the main window

## 5.5 Future work: possible improvements

### 5.5.1 Writing new procedures

The chosen architecture enables to write new procedures. A fast way to extend the tool is to implement some common used procedures:

New clustering procedures enable to have clusters that make more sense. Because the coloration of the graph is directly linked to the partition produced by the clustering procedure, this will improve the readability of the graphs.

A second type of procedures is the dimension reduction procedure. A lot of procedures already exist, and some of them might be better than others, depending on the used dataset. The user will have to try different procedures and chose the

most relevant for his problem.

The last type of procedure is the $n$-dimensional plotting procedure. If parallel coordinates graph is one of the easiest way to represent n-dimensional data, it doesn't suite all kind of data. For example binary spaces (like in decision space for the knapsack problem) represented with parallels coordinates graph are not easy to read. For this application a new procedure can be written.

### 5.5.2 Modifying the graph appearance

In this first version, the appearance of the graph is not very elaborate. Default settings were used. A lot of things could be improved (format of the numbers on the axis, position of the legend box, ...)

The main window is splitted in four sub-graph areas. Even on big screen, the graphs might be difficult to read with large dataset. The tool could provide zooming features and could also offer the possibility to see one of the four graphs in a full screen mode.

### 5.5.3 Adding interaction

In the first version, the user's interaction is limited to the use of the modal dialog. The user can't click in the graph zone. Making the graph areas selectable could improve the user's experience. A selected point has to be identified in the four graphs (with for example a different color or a bigger size).

Such a behavior often comes with grouping features. Like in many vector-editing tools, it could be possible to perform Ctrl click and Shift click. It would be very interesting if the groups created by the user could become a new partition of the population. This would permit to transform the groups made by the user into clusters of points that get their own coloration and that can be displayed or not.

### 5.5.4 Review the user interface

Giving the tool a multi-window support with its owns menu-bar will improve the user's experience. That will enable to manage different datasets and to work in parallel in different windows.

When the tool becomes bigger with a lot of procedures (clustering, plotting), the modal dialog windows might need to be redesigned. A single big combo-box for the selection of the procedures might be difficult to use. It would also be nice if the tool was able to make pre-selection, by indicating which procedure provides

better results given the loaded datasets (the metrics of the 2 previous chapters can be used).

# Chapter 6

# Conclusion and future work

## 6.1 Conclusion

In the context of multiobjective optimization search algorithms provide a set of optimal solutions called Pareto front. Each solution can be modelled with an objective vector and a decision vector. The decision maker has to choose one of theses solutions, therefore it is important that he get a good overview of the set of solutions. The goal of this work was to provide the decision maker a visualization tool that represents a set of solutions.

Two problems come with a set of Pareto solutions: the coordinates vectors lie in a high dimensional space and there is a large number of points in the set. Before the implementation of the visualizing tools, two concepts where studied to deal with the problems.

The dimension reduction is a method to map a high dimensional space onto a two-dimensional one which is easier to represent. Drawbacks of such procedures: information is lost. A lot of dimension reduction techniques already exist, we need to select the more suitable in our context. Three different metrics were proposed, in order to measure how good a dimension reduction method is, in regard to the properties we want to preserve: the distance and dominance relation. With the metrics two standard, very often used dimension reduction techniques were compared: Principal Component Analysis and Kohonen's self-organizing maps. There is no big difference between the two methods, but the results for Principal Component Analysis are a little better.

Clustering procedures group similar points together. They reduce the number of points that the decision maker has to consider. Similarity groups are interesting in the objective space and in the decision space. The structure of these two spaces is not the same, and performing a clustering procedure simultaneously in the two

spaces could be an interesting idea. We proposed an extension of the $k$-means algorithm, in order to be able to work in two spaces simultaneously. We verify that the algorithm works as expected depending on which space is considered (objective space only, decision space only and both spaces simultaneously).

The realized Java application is a basis tool to visualize a Pareto front including dimension reduction and clustering procedures. The application is usable and implements some of the procedures described in this report. It can be considered as a first version that provides a good architecture for future developments.

## 6.2 Future work

In the previous chapter some ideas are already given concerning the future work. For the dimension reduction and clustering procedure the main idea is to get more results. The tests have to be continued with other datasets and other procedures. The proposed methods and metrics can be reapplied. This will enable to go more in details. For the visualization tool, a lot of improvements are proposed to extend this first basic version. The main priority is to implement other procedures (clustering, plotting), in order to extend the features of the tool.

More generally, other visualization concepts could be taken into consideration. This work stays very close to the numerical values of the coordinates vectors of the points. The graphs represent the values of the points with a lossless or reduced representation. It is possible to use other techniques like interaction, virtual reality or symbolization with pictures. These techniques provide a totally different visualization's approach for the set of points and can give the decision maker another view to the problem he is considering. These alternative representations can help him for the comparison of two solutions, the evaluation of one solution and the final decision.

# Bibliography


[1] Stefan Bleuler, Marco Laumanns, Lothar Thiele, and Eckart Zitzler. PISA — a platform and programming language independent interface for search algorithms. In Carlos M. Fonseca, Peter J. Fleming, Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele, editors, *Evolutionary Multi-Criterion Optimization (EMO 2003)*, Lecture Notes in Computer Science, pages 494 – 508, Berlin, 2003. Springer.

[2] M.A. Carreira-Perpinan. A review of dimension reduction techniques. *Technique Report, CS-96*, 9, 1996.

[3] A. Inselberg. The plane with parallel coordinates. *The Visual Computer*, 1(4):69–91, 1985.

[4] S.C. Johnson. Hierarchical clustering schemes. *Psychometrika*, 32(3):241–254, 1967.

[5] I.T. Jolliffe. *Principal component analysis*. Springer New York, 2002.

[6] T. Kohonen. *Self-Organizing Maps*. Springer, 2001.

[7] P. Korhonen and J. Wallenius. Visualization in the multiple objective decision-making framework. In *Multiobjective Optimization: Interactive and Evolutionary Approaches*, pages 195–212. Springer, 2008.

[8] Z. Kutalik, JS Beckmann, and S. Bergmann. A modular approach for integrative analysis of large-scale gene-expression and drug-response data. *Nat Biotechnol*, 26(5):531–9, 2008.

[9] A. V. Lotov and K. Miettinen. Visualizing the pareto frontier. In *Multiobjective Optimization: Interactive and Evolutionary Approaches*, pages 213–243. Springer, 2008.

[10] J. B. Macqueen. Some methods of classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297, 1967.

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

# Appendix A

# User Guide

## A.1 Introduction

HDPlot is a visualizing tool for High dimensional data. It is designed for multiobjective optimization dataset. These data have the particularity to live simultaneously in an objective space and a decision space. Both spaces are high dimensional.

The Goal of HDPlot is to provide a nice way to represent these data in different graphs (non destructive and reduced in a 2 dimension representation). HDPlot is designed to be used with data provided by multiobjective optimizers compatible with the PISA platform.

### A.1.1 System requirement

HDPlot is programmed in Java, so you need a compatible operating system with a recent version of Java. It works perfectly fine on Ubuntu 8.04 with java 1.6.0_07

### A.1.2 File format

The input file is a text file which contains the values. Each line represents a point. The first column can be an index, the next columns are the coordinates in the objective space and the remaining part stands for the coordinates in the decision space. Each value is separated with a blank space. You can see an example of such a file in Figure A.1.

The illustrations of this guide are made with datasets derived from knapsack problem. The 2 files are provided:

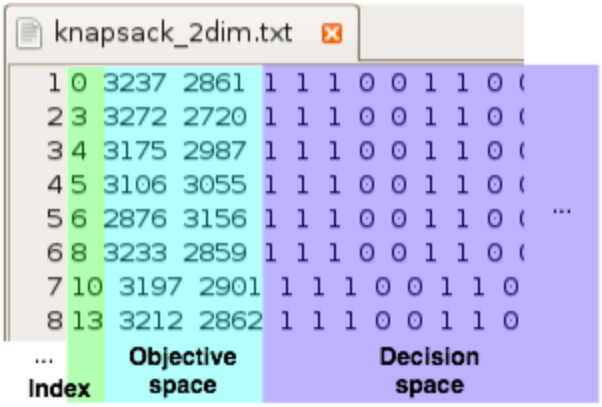


Figure A.1: Screeshot of knapsack_2dim.txt

**knapsack_2dim.txt:** a 2 dimensions problem (with index in the first column)

**knapsack_4dim.txt:** a 4 dimensions problem (without index column)

## A.2 Main window

When you open HDPlot you see main window (Figure A.2)

**Command bar:** like in a menu bar, you find here some buttons that enable you to interact with the program.

**Graphs zone:** this is the main part of the window. In this area, graphs are plotted.

**Status bar:** provides you informations about the current status of the application (when you have done an action, you can get confirmation that everything worked fine).

In the graph zone four graphs are plotted in $2 \times 2$ rectangle grid (Figure A.3). The top line is for the objective space, the bottom line is for the decision space. The left-hand column is for $n$-dimensional reprensation (non destructive), the right-hand column is for two-dimensional reprensation (with a dimension reduction procedure).

The next sections describe how to use the different buttons available in the command bar (Figure A.4). You will learn:

- How to open a file. (Open in Figure A.4, see Section A.3.1)

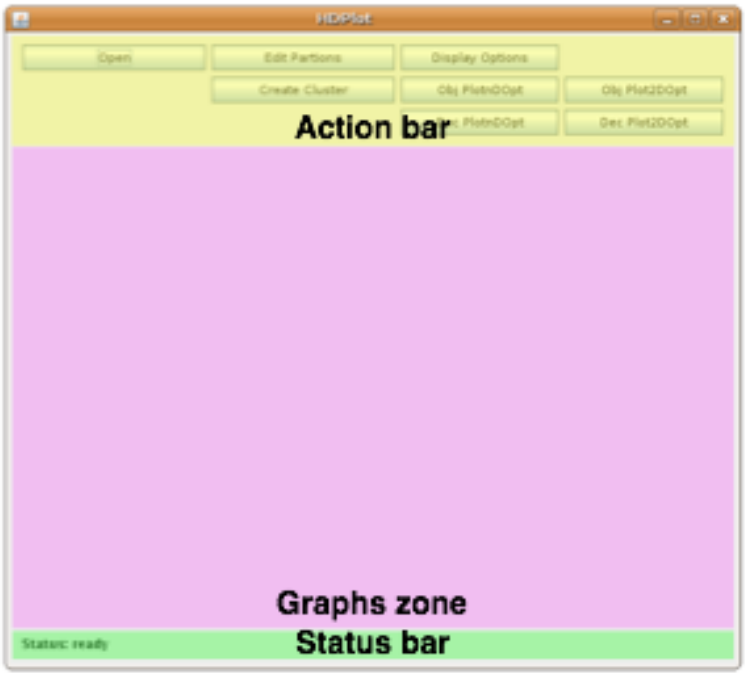


Figure A.2: Main windows of the HDPlot

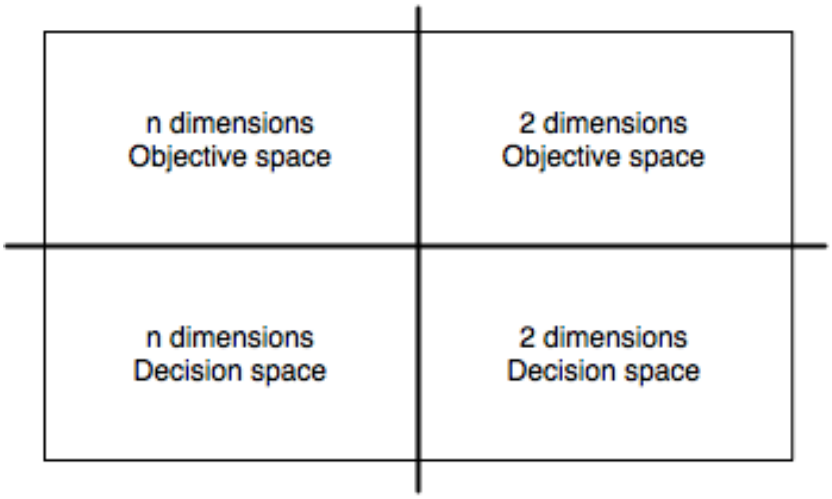


Figure A.3: Organisation of the Graphs zone

- How to handle partitions. (Partition in Figure A.4, see sections A.4.1 and A.4.2)

- How to modify the graphs. (Display options in Figure A.4, see sections A.5.1, A.5.2 and A.5.3)


Figure A.4: Command bar of the main window

## A.3 Opening a file

### A.3.1 Open dialog

The first step you have to do in order to use HDPlot, is to open a dataset file. You open the ”opening dialog” by clicking on the ”open” button in the command bar.

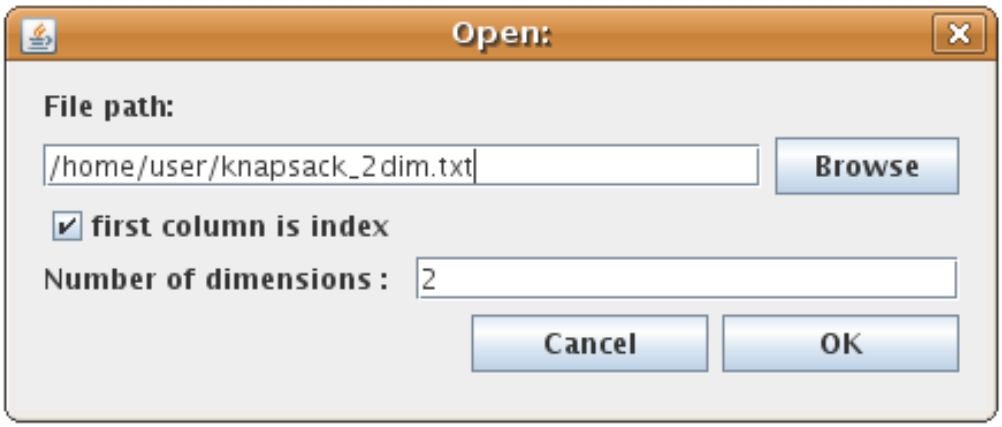


Figure A.5: Open dialog

The dialog (Figure A.5) is quite simple. You need to provide:

- The file path. If you want, you can use the ”Browse” button in order to get the file path automatically.

- If your file contains an index column, you need to have the checkbox ”first column is index” checked. Otherwise (the first column is the first coordinate in the objective space) just uncheck the checkbox.
- You also need to provide the number of dimensions of the objective space.

## A.4 Managing the partitions

An important concept of HDPlot is the management of partitions. A partition is a division of the whole population of points into non-overlapping groups of points. The partitions are afterwards used for the display options (coloration, etc).

When you load a dataset, a first partition called ”whole population” is created. This partition consists of a single group a points ”all individuals” that contains all points. HDPlot enables you to manage the partitions and to create new ones.

### A.4.1 Edit partitions dialog

To manage the partitions, click on the ”Edit Partition” button. This opens a quite simple dialog (Figure A.6)

In the listbox you see the current partitions. In order to add a new partition you can click on the ”Add” button. This open the dialog described in Section A.4.2. In order to remove a partition, just select it in the listbox and click on the ”Remove” button. When you are finished, just click on the ”Done” button.

### A.4.2 Create a partition (create cluster dialog)

The partitions are created with clustering procedures. You can open the Create cluster dialog either from main window (click on the ”create cluster” button) or form the edit partition dialog (click on the ”add” button)

At the top of the dialog you can choose one of the available clustering procedures. Depending on the procedure, you need to provide suitable settings. The different procedures are discribed in the next sections. You also need to provide a name to the new partition. Validate your choices by clicking on the ”Ok” button.

#### A.4.2.1 Split with a threshold in one dimension

This clustering procedure creates 2 clusters. You select a dimension (either one of the objectives in the objective space, or one of the coordinates of the decision space)


Figure A.6: Edit partition dialog

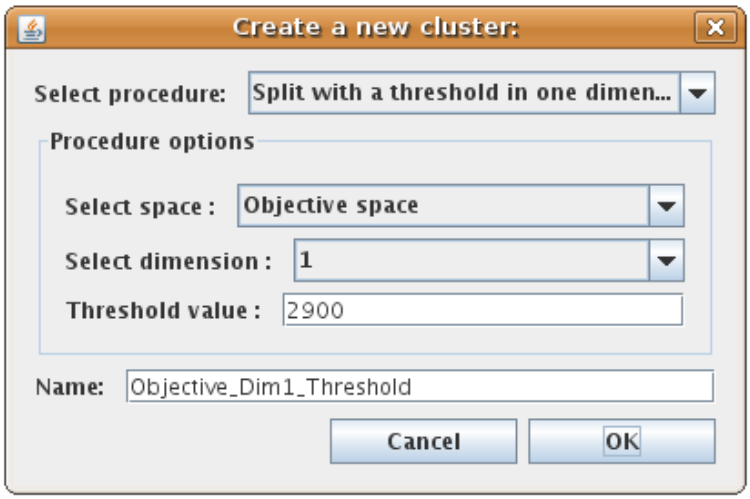


Figure A.7: Create cluster dialog

and a threshold. For the selected dimension, all points that are below the threshold value go in the first cluster and all the other points go in the second cluster.

The available options are visible in the Figure A.7. The usage is quite straightforward. The first combo-box enables you to choose if you want to work with the objective space or the decision space. The second combo-box enables you to select the dimension. You can set the threshold in the text-field.

#### A.4.2.2 $k$-means

This clustering procedure is a implementation of the extended $k$-means procedures described in the Section 4.3. You need to provide a value for $k$ (the number of cluster) and $\alpha$ (balance between the two spaces).

#### A.4.2.3 Random cluster

*This clustering procedure is designed for test.* You can set the number of cluster that should be created. The points are assigned randomly to each cluster. The number of points in each cluster does not exceed the selected maximal value.

#### A.4.2.4 Split cluster

*This clustering procedure is designed for test.* 2 clusters are created. The first cluster contains the first half of the population. The second cluster contains the other half.

## A.5 Modifying the graphs

### A.5.1 Display options dialog

The display options dialog (Figure A.8) allows you to choose the points you want to display and to enable coloration or not. You need to work with one of the partition you previously created (see Section A.4.2).

The dialog window is composed as following:

- At the top, you have a combo-box which displays all available partitions. Select the one you want to work with.
- In the center of the window, you see all the clusters of this partition in a list-box. You need to select those that you want to display. As usual, you can use

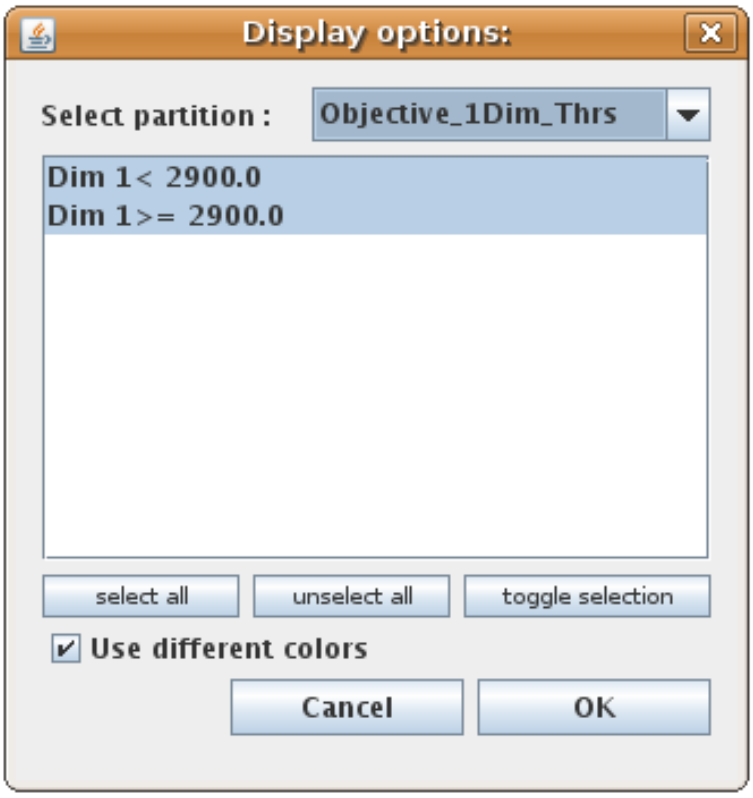


Figure A.8: Display options dialog

the Ctrl and the Shift keys in order to extend or reduce the selection. You can also use the small buttons ("select all", "unselect all" and "toggle selection"). At the end, only clusters that are selected will be displayed.

- Finally a check-box enables you to use colors: If you uncheck it, all points will be black. Otherwise, each cluster will get a different color (with the following limitation: the color palette only contains 16 colors. The seventeenth cluster gets the same color as the first one)

When you are finished, just click on "Ok".

### A.5.2 $n$ dimentions graph options dialogs

You can change the representation of the $n$-dimensional objective space graph (resp. decision space graph) by clicking on the "Obj PlotnDOpt" button (resp. the "Dec PlotnDOpt" button). The dialog Figure A.9 appears.

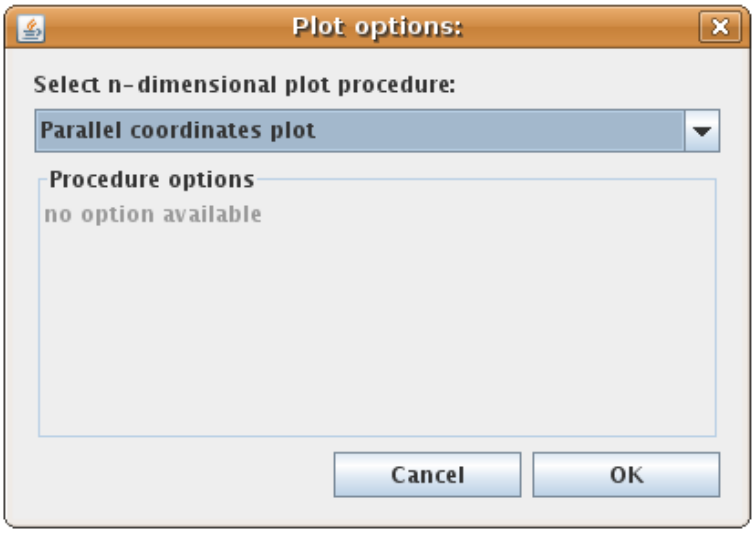


Figure A.9: $n$-dimensional plot option dialog

At the top, you choose one of the available plotting procedures. They are described in the next sections.

#### A.5.2.1 Parallel coordinates

This plotting procedure produces the standard parallel coordinates graph. No customization options are available.

### A.5.3 Two dimentions graph options dialogs

Each space can be displayed on a 2 dimensions graph, by using a dimension reduction procedure. The option dialog (Figure A.10) looks quite similar to the $n$-dimensional one (Section A.5.2). You open it with the "Obj Plot2DOpt" or "Dec Plot2DOpt" button.

Select one of the available procedures in the combo-box and set the corresponding settings if needed. The dimension reduction procedures are described in the next sections.

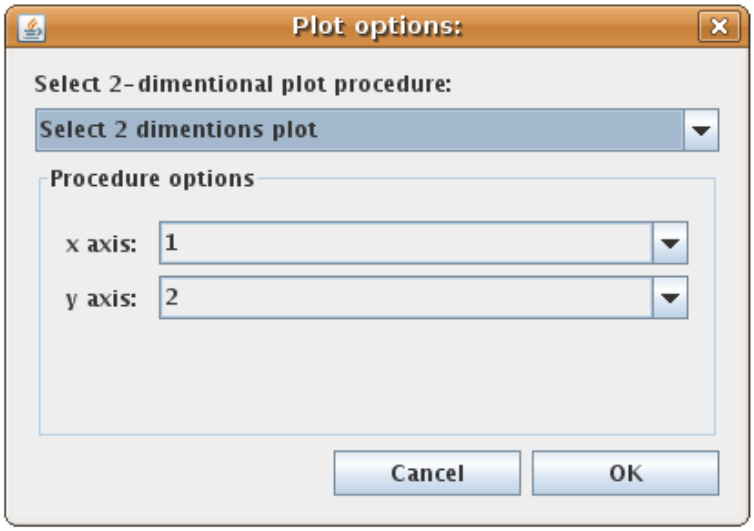


Figure A.10: two-dimensional plot option dialog

#### A.5.3.1 Principal Components Analysis on correlations

This dimension reduction procedure uses Principal Components Analysis (see Section 3.2.2). No customization options are available.

#### A.5.3.2 Select 2 dimensions

This dimension reduction procedure is very simple. You just provide the dimension you want to use on the x-axis and the one you want to use on the y-axis.

## A.6 Screenshots

The following sections give you examples of what can be done with HDPlot.

### A.6.1 Screenshot 1

See Figure A.11. A simple example with knapsack_2dim.txt.

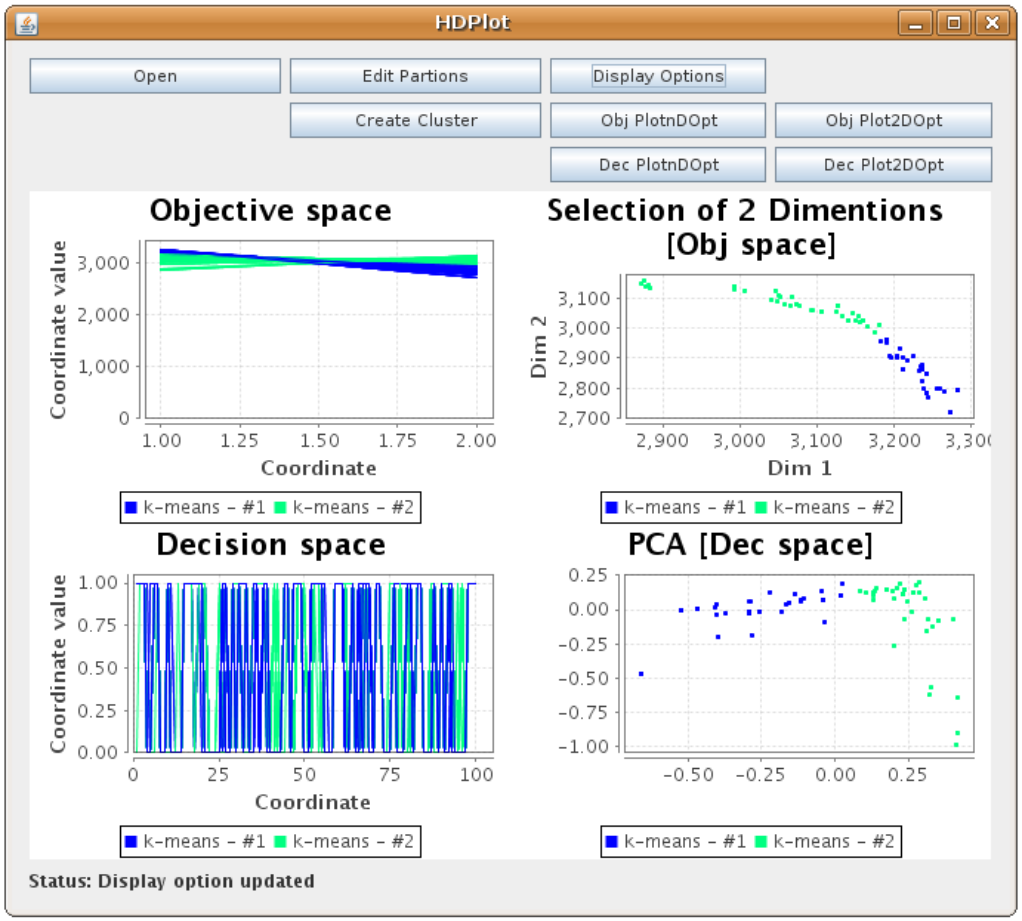


Figure A.11: Screenshot: The main window displaying knapsack_2dim.txt with 2 clusters

### A.6.2 Screenshot 2

See Figure A.12. A four dimensions example. To disable color see A.5.1.

### A.6.3 Screenshot 3

See Figure A.13. The dataset used here is the same as in the screenshot 1 (A.11), but a new partion has been created and used for coloration. For the creation of partition see A.4.1. For display options see A.5.1.

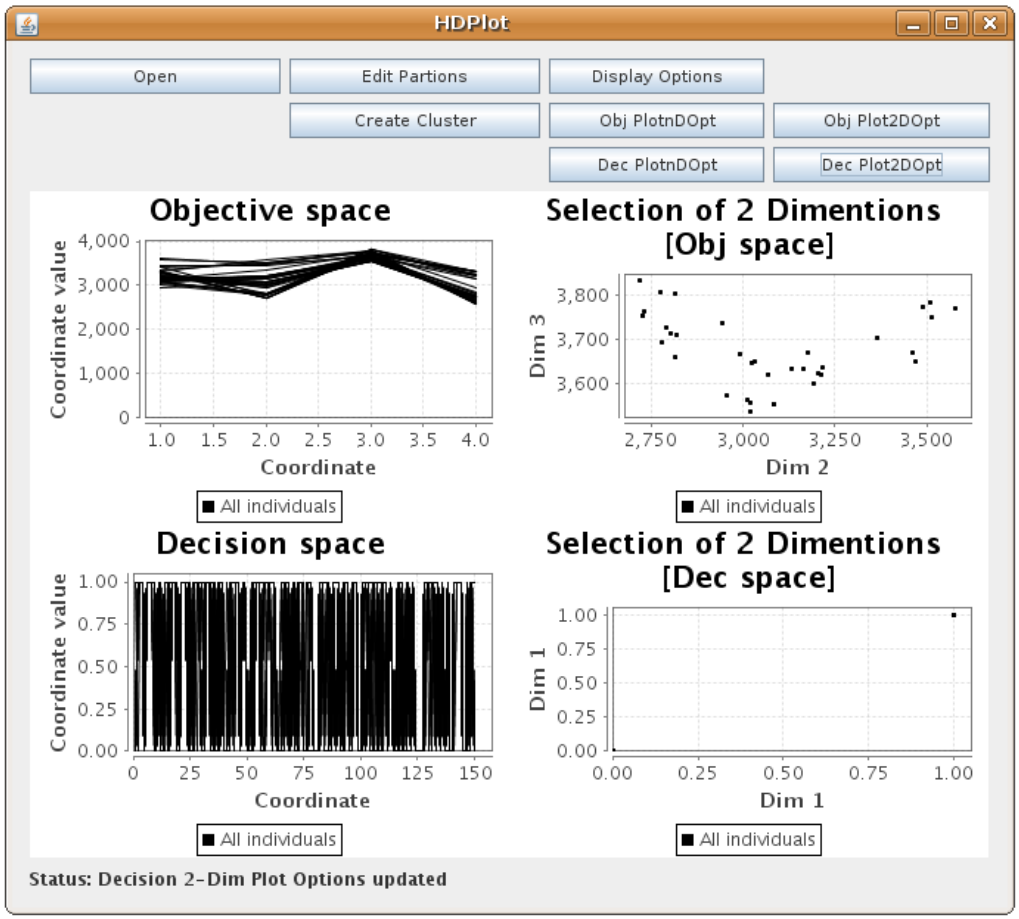


Figure A.12: Screenshot: The main window displaying knapsack_4dim.txt without color

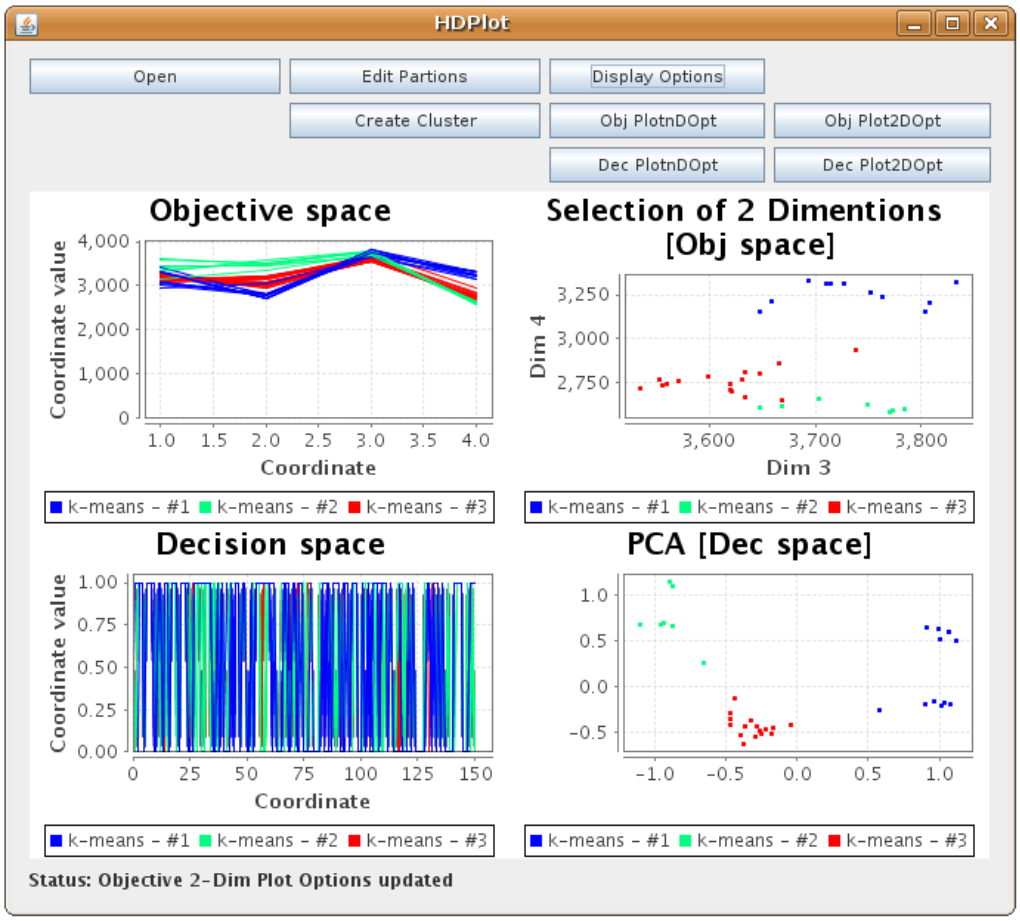


Figure A.13: Screenshot: The main window displaying knapsack_4dim.txt with a $k$-means clustering procedure

# Appendix B

# Developer Guide

This guide explains the general organization of the code in the first part and provides tutorials for programmers who want to extend the program in the second part.

HDPlot was programmed with NetBeans[1] (but it also can be opened in Eclipse[2]). It compiles with java version 1.6.

The javadoc documentation of the code also provides detailed informations.

## B.1 Overview

### B.1.1 The different packages

#### B.1.1.1 struct

This package contains 4 classes that allow you to instantiate a representation of the data. The `Coordinates` store $n$ doubles. Each `Individual` has got 2 `Coordinates` (one for each space). The `Individuals` are grouped in an `IndividualSet` (e.g the whole population is stored in a `IndividualSet`). A `Partition` is a set of `IndividualSet` with some control methods in order to maintain the coherence of the partition.

Technically the Classes are very simple. They store the content in private attributes and provide setter and getter methods. These methods also check the coherency of the data. For sets, the `java.util.Vector` (for his simplicity) is used.

[1] http://www.netbeans.org/
[2] http://www.eclipse.org/

#### B.1.1.2 textio

This reusable package provides classes so as to easily read and write text files.

#### B.1.1.3 cluster

This package groups all classes needed for the different clustering procedures. It provides an abstract parent class `AbstractCluster` which must be extended by the subclasses. For each procedure a subclass has to be created (example: `SingleCluster`, `Threshold1Dim`, etc). If the procedure needs user inputs, a complementary class which controls the user interface must be added. (example: `Threshold1DimOpt`)

#### B.1.1.4 plotnD

This package contains all classes needed for the $n$-dimensional plotting procedure. It works like the `cluster` package, but instead of an abstract class which is extended by each procedure, the package uses an interface `PlotnDInterface` that is implemented in all procedures.

#### B.1.1.5 plot2D

This package is the same as the `plotnD` package but for the dimension reduction procedures.

#### B.1.1.6 gui

This package groups all modal dialog windows. Each window implements `ModalDialogInterface`.

The package also provides an interface (`OptInterface`) which must be implemented by all classes that control the user interface in the procedure packages (`cluster`, `plotnD` and `plot2D`). Additionally the package provides a class `OptEmpty`, that implements this interface by displaying a simple "no options available". This class has to be used by all procedures that do not need any user input.

#### B.1.1.7 hdplot

This is the main package of the application. It contains the rest of the classes.

- `Main` is the class that is lunched at startup.
- `JFMainWin` is the main window. This class is linked to the `JFMainWinState` class. (The code is spitted in 2 classes so as to improve readability of the code). The code in both classes links all the elements of the application together.
- `ProcedureInterface` must be implemented by all the procedures (clustering and plotting) mentioned above (in packages `cluster`, `plotnD` and `plot2D`).
- `DisplayInJPanelInterface` must be implemented in the plotting procedure in order to display the result in a `JPanel` in the main window.
- `JFreeChartPlot` is an abstract class that can be used by the plotting procedures if they want to use the JFreeChart library. The class pools code that can be used by different plotting procedures and gives a default behavior. Of course plotting procedures might be written from scratch. `JFreeChartPlot` starts the implementation of `DisplayInJPanelInterface`.
- `SettingsColorServer` and `SettingsProcedure` provide some default settings that are used at different points of the application.

### B.1.2 Extending the software

HDPlot is ready to be extended. It is very easy to write and implement a new procedure. It has to implement `hdplot.ProcedureInterface`.

- A clustering procedure belongs to the `cluster` package and extends `cluster.AbstractCluster`
- A $n$-dimensional plotting procedure belongs to the `plotnD` package and implements `plotnD.PlotnDInterface`. It can be started from scratch or extend the abstract class `hdplot.JFreeChartPlot` in order to reuse some useful code and the JFreeChart library.
- A two-dimensional plotting procedure belongs to the `plot2D` package and implements `plot2D.Plot2DInterface`. It also can extend `hdplot.JFreeChartPlot` if needed.

As the goal of a plotting procedure is to be displayed in the main windows, it must also implement `DisplayInJPanelInterface`.

Some procedures also need user inputs. The user interface in the modal dialogs is ready to be automatically modified when the user selects a procedure. It means that the content of a JPanel must be updated with new SWING components. Such

behavior is coded in classes that implements `gui.OptInterface`. By default the `gui.OptEmpty` class can always be used.

Writing a new procedure consists in writing the procedure class and the option class. The detection of the procedure is not automatically. The combo-boxes, which enable the user to select a procedure, are based on the content of vectors provided by the `hdplot.SettingsProcedure` class. When you have written a new procedure, do not forget to add it to the corresponding vector.

The tutorials in the next sections give step-by-step explanations to extend the program.

## B.2 Tutorials

### B.2.1 How to create a new clustering procedure ?

This tutorial describes how to create a clustering procedure. Code extracts and names are taken from the procedure `cluster.Threshold1Dim`.

#### B.2.1.1 Step 1: create a new class

The goal of a clustering procedure class is to provide a new `struc.Partition`. If some values are needed in order to create the partition, they are stored as private attributes. This means that you might also need to provide setter and getter methods.

All the clustering procedures need to provide a set of common public methods in order to interact with the rest of the program. They are prepared as abstract methods of the `AbstractCluster` class and described in the next steps.

In order to keep the organization of the code, all the clustering procedures belong to the `cluster` package.

#### B.2.1.2 Step 2: extend AbstractCluster

All the clustering procedures need to extend `AbstractCluster`.
Add "`extends AbstractCluster`" in the class declaration.

This defines obligatory functions that must be implemented.

#### B.2.1.3 Step 3: constructor method

The constructor sets a link to the state of the main window (`JFMainWinState`). This is done in the constructor of `AbstractCluster`. You need to call the `super()` method. The shortest constructor method looks like this:

```
public Threshold1Dim(JFMainWinState theStateInit) {
    super(theStateInit);
}
```

You can complete it with some other attribute initializations. The state of the main window is accessible through the protected `theState` attribute.

#### B.2.1.4 Step 4: getDisplayName() method

This method must return the name of the procedure that is displayed in the program. It is used in the selection combo-box in the "create cluster" modal dialog.

#### B.2.1.5 Step 5: getOpt() method

This method must return an instance of the class that is used in order to modify the user interface. If your procedure does not need any user input you are fine with the lines:

```
public gui.OptInterface getOpt() {
    return (new gui.OptEmpty());
}
```

Otherwise you need to create a class that will create the user interface (for example: `Threshold1DimOpt`). See the tutorials B.2.2 and B.2.3. After that you can use it in this method:

```
public gui.OptInterface getOpt() {
    Threshold1DimOpt opt = new Threshold1DimOpt();
    opt.setBackLink(this);
    return (opt);
}
```

Remark the usage of the `setBackLink()` method in order to link the option class with the procedure class.

#### B.2.1.6 Step 6: generateNewPartition() method

This is the core method of the clustering procedure. It is called when the user validates the modal dialog. It must return the new created partition.

#### B.2.1.7 Step 7: add the procedure to the combo-box

This is done in the `hdplot.SettingsProcedure` class. You need to edit the static `clusterProcedureVector()` method. Instances of the clustering procedures classes are added to the vector v. Just add a new element to the vector (at the desired position in the combo-box):

```
v.add(new Threshold1Dim(currState));
```

### B.2.2 How to create a option JPanel from scratch ?

An option class creates a procedure-specific user interface. It creates the content of a JPanel with SWING components, controls the accuracy of the user input and sets back attributes of the corresponding procedure.

#### B.2.2.1 Step 1: create the class

By convention the option class is in the same package as its corresponding procedure. Its name is the same as the procedure class with an "Opt" suffix (for example: `Threshold1DimOpt`). All option classes must implement `gui.OptInterface`. The methods are described in the next steps.

#### B.2.2.2 Step 2: getDisplayName() method

This method must return the name of the option class that is displayed in the program. *Currently it is not used.*

#### B.2.2.3 Step 3: setBackLink() method

This method creates a link between the option class and the corresponding procedure class. The parameter `o` of the function is a `hdplot.ProcedureInterface` (as defined in `gui.OptInterface`), but in order to be more specific, the object has to be cast into the corresponding procedure class. The `o` parameter should be copied in a private attribute (conventionally it is called proc).

For example the `setBackLink()` method of `cluster.Threshold1DimOpt` :

```
public void setBackLink(hdplot.ProcedureInterface o) {
    this.proc = (Threshold1Dim)(o);
}
```

#### B.2.2.4 Step 4: initCntComponents() method

This method creates the SWING user interface. The components must be added to the JPanel passed by the parameter `jPCnt`.

#### B.2.2.5 Step 5: storeSettings() method

This method is called when the user validate the dialog. It is used to check if the inputs are correct or not. If everything is all right, the method must set the input into the procedure class and return true, otherwise it must warn the user and return false. In that case, the dialog stays open.

### B.2.3 How to create a option JPanel with Swing GUI Builder ?

Tested with NetBeans 6.5[3]

If this method is often used, the creation of a new NetBeans Template could be considered.

#### B.2.3.1 Step 1: Create a new JPanel

Create a new JPanel Form (right click on the source package > New > New JPanel Form... )

#### B.2.3.2 Step 2: Create normally your GUI with the tool

You can use the tool as usual and even provide code for the events of the component you included. You might come back to do other modifications. When done, move to the "Source section" (if you come back later on you will need to redo step 6 [page 68]).

[3]Swing GUI Builder is included in NetBeans (formerly Project Matisse) http://www.netbeans.org/features/java/swing.html

#### B.2.3.3 Step 3: change the Class declaration

In the Class declaration change "`extends javax.swing.JPanel`" by "`implements gui.OptInterface`"

#### B.2.3.4 Step 4: Comment the initComponents() method

You must surround the "Generated Code" section so that it becomes a comment. For example, you have:

```
/** This method is called from within the constructor to
 * initialize the form.
 * WARNING: Do NOT modify this code. The content of this method is
 * always regenerated by the Form Editor.
 */
@SuppressWarnings("unchecked")
[Generated Code]
```

You can do:

```
/** This method is called from within the constructor to
 * initialize the form.
 * WARNING: Do NOT modify this code. The content of this method is
 * always regenerated by the Form Editor.
 */

/*
@SuppressWarnings("unchecked")
[Generated Code]
 */
```

Be careful to surround just the `initComponents()` method. You should not comment the event handler methods (like `jButton1ActionPerformed(... evt)`). Be also careful to close the "`/**` Warning block". You should not merge the warning block with the comment block of the `initComponents()` method. If you merge both blocks you might get a star in front of the event handler methods.

#### B.2.3.5 Step 5: remove the constructor method

If needed you can just modify it. The point here is to remove the call of the `initComponents()` method because it does not exist anymore (step 4 [page 67]).

#### B.2.3.6 Step 6: create the initCntComponents(JPanel jPCnt) method

This is very simple: you need to copy and modify the auto generated `initComponents()` function. You have just commented this function in step 4 [page 67]. Copy and past it in the `initCntComponents(JPanel jPCnt)` function. Replace all the "`this`" occurences with "`jPCnt`" (because this is not a `JPanel` any longer (step 3 [page 67]) and you are interested in changing the passed variable `jPCnt`) If your code is very long (which of course depends on the complexity of your GUI), you can use a "find and replace" tool.

#### B.2.3.7 Step 7: Create the other medods

The `gui.OptInterface` Interface needs more methods, implement them like you do without the Swing GUI Builder tool (tutorial B.2.2).

### B.2.4 How to create a $n$-dimensional plotting procedure ?

The $n$-dimensional plotting procedure is very similar to the clustering procedure. The tutorial B.2.1 can be used with the following remarks:

- A new plotting procedure must be placed in the `plotnD` package and must implement `plotnD.PlotnDInterface`. The functions that have to be implemented are very similar.
- Instead of the `generateNewPartition()` method, the principal method is `plotInJPanel()`. This method must display the plotted graph into a JPanel passed as parameter.
- In order to add the new created procedure into the combo-box, the static method `plotnDProcedureVector()` of `hdplot.SettingsProcedure` has to be modified.

### B.2.5 How to create a two-dimensional plotting procedure ?

The 2-dimensional plotting procedure is very similar to the clustering procedure. The tutorial B.2.1 can be used with the following remarks:

- A new plotting procedure must be placed in the `plot2D` package and must implement `plot2D.Plot2DInterface`. The functions that have to be implemented are very similar.

- Instead of the `generateNewPartition()` method, the principal method is `plotInJPanel()`. This method must display the plotted graph into a JPanel passed as parameter.
- A 2-dimensional plotting procedure contains a dimension reduction procedure. The method `train()` should set private attributes that are used for dimension reduction (if needed). This method is called after the user validates his choice and before the `plotInJPanel()` is called.
- In order to add the new created procedure into the combo-box, the static method `plot2DProcedureVector()` of `hdplot.SettingsProcedure` has to be modified.