JOURNAL OF MULTI-CRITERIA DECISION ANALYSIS
J. Multi-Crit. Decis. Anal. (2012)
Published online in Wiley Online Library
(wileyonlinelibrary.com) DOI: 10.1002/mcda.1477

Pareto-Set Analysis: Biobjective Clustering in Decision and Objective Spaces

TAMARA ULRICH*
Computer Engineering and Networks Laboratory, ETH Zurich, Zurich, Switzerland

ABSTRACT

Multiobjective problems usually contain conflicting objectives. Therefore, there is no single best solution but a set of solutions that represent different tradeoffs between these objectives. Knowledge of this front can help in understanding the optimization problem better, as promising designs can be identified, and it can be seen what the achievable tradeoffs between the objective values are. Although for real-world problems, this interpretation of the front is usually not straightforward.

This paper proposes a method to help the decision maker by clustering a given set of tradeoff solutions. It does so by extending the standard approach of clustering the solutions in objective space, such that it finds clusters that are compact and well separated both in decision space and in objective space. It is not the goal of the method to provide the decision maker with a single preferred solution. Instead, it helps the decision maker by structuring the tradeoff solutions such that he or she can learn about the problem. More precisely, a good clustering of the tradeoff solutions both in decision space and in objective space elicits information from the front about what design types lead to what regions in objective space. The novelty of the presented approach over existing work is its general nature, as it does not require the identification of distinct design variables or feature vectors. Instead, the proposed method only requires that a distance measure between a given pair of solutions can be calculated both in decision space and in objective space.

As good clusters in decision space do not necessarily correspond to good clusters in objective space, we formulate this clustering problem as a biobjective optimization problem and propose PAN, a multiobjective evolutionary algorithm, to generate promising partitionings. Tests on artificial datasets are used to identify a suitable representation and a suitable partitioning goodness measure for PAN. Results from applying PAN to a knapsack problem and a bridge construction problem show that PAN is able to find multiple tradeoffs between good clustering in decision space and in objective space. Copyright © 2012 John Wiley & Sons, Ltd.

KEY WORDS: Pareto-set analysis; clustering; evolutionary multiobjective optimization; design principles

1. INTRODUCTION

When optimizing complex systems, there are often many nonlinear conflicting objectives that have to be considered. One way to approach such a problem is to generate the set of Pareto-optimal solutions – or to approximate it, if the problem is too complex. The advantage is that the process of selecting a preferred solution is postponed, thereby giving the user the possibility to extract information about the problem from the Pareto-optimal set. The drawback is that selecting a single preferred solution can be difficult, as the Pareto-optimal set may contain a large number of solutions, which are difficult to compare if there are more than two or three objectives and if there is a complex decision space representation. Moreover, it is not quite clear how information about the problem can be gained from the approximated set. In this paper, we would like to group similar solutions in order to help the decision maker learn about the problem and to select a preferred solution.

In real-world problems, engineers who optimize such problems are not only interested in the objective values of the found solutions but also in their structure, that is, how the designs look like. For example, considering a bridge optimization problem, the engineer might not only be interested in the cost of the bridge and the load it can carry but also in the shape of the bridge (e.g. whether it is a cantilever bridge, a cable-stayed bridge or a suspension bridge, how many pillars it has or how many arches it contains). Therefore, when we try to find groups of similar solutions, we would like to find groups that

*Correspondence to: Computer Engineering and Networks Laboratory, ETH Zurich, 8092 Zurich, Switzerland. E-mail: tamara.ulrich@tik.ee.ethz.ch

Received 31 August 2011
Accepted 17 June 2012

are similar both in objective values as well as in their structure. With such a clustering, an engineer can draw certain conclusions about which types of designs lead to what regions in objective space.

Also, when optimizing real-world problems, the interpretation of the achieved solutions might be very time consuming, because the representation of each solution is very complex. Assume for example in the bridge construction problem that each bridge can be built from different materials such as concrete, steel or other metals. For concrete, the size and the shape of the steel reinforcing could also be optimized. In addition, there might be different steel types such as steel bars or steel wires of arbitrary thickness or even steel arcs of arbitrary shape. The more detailed the model to be optimized becomes, the more time it takes to inspect the solutions returned by the optimization algorithm. Here, it is useful to have an automatic method to structure the solutions, such that the engineer only has to look at a few of them and knows that the remaining solutions are of a similar structure with similar objective values.

This paper makes the following contributions:

- It formulates the biobjective optimization problem of clustering the solutions in objective space and in decision space simultaneously.
- It selects suitable cluster validity indices that can be used as the objective functions.
- It extensively tests several evolutionary algorithm representations and several cluster validity indices for their use during biobjective clustering optimization.
- It applies its findings to the fronts of a biobjective knapsack problem and a real-world bridge construction problem.

After formally stating the optimization problem in Section 2 and discussing related work in Section 3, we describe the details of our evolutionary algorithm in Section 4, including a discussion of different representations and the use of validity indices to measure partitioning goodness. To select a validity index and a suitable representation, different combinations are tested in Section 5 on artificial datasets where the optimal partitionings are known in advance. Finally in Section 6, the algorithm is compared with the standard approach of repeatedly applying the well-known *k*-medoids clustering algorithm with different number of clusters in order to validate the multiobjective approach. The proposed method is then applied to the Pareto-set approximations of a knapsack problem and of a bridge optimization problem to qualitatively evaluate its results.

2. PROBLEM SETTING

Consider a multiobjective optimization problem with a decision space X and an objective space $Z \subseteq \mathbb{R}^m = \{f(x) | x \in X\}$, where $f: X \rightarrow Z$ denotes a mapping from the decision space to the objective space with m objective functions $f = \{f_1, \ldots, f_m\}$. An element $x \in X$ of the decision space is also named a solution. Although the objective space is a real-valued space, we make no assumptions about the structure of the decision space. In particular, we do not require the decision space to be a Euclidean space, and we also do not require the decision space to be spanned by a predefined set of decision variables that can take a certain set of values. Instead, we only assume that we are given a distance measure on solution pairs, that is, $d_D : X^2 \rightarrow \mathbb{R}$, where $d_D(x_1, x_2) \in \mathbb{R}$ denotes the structural distance between the two solutions x_1 and x_2.

We will also need a distance measure in objective space. As the objective space is a real-valued metric space, we choose Euclidean distance, that is, $d_O : X^2 \rightarrow \mathbb{R}$, with $d_O(x_1, x_2) = \sqrt{\sum_{i=1}^{m} (f_i(x_1) - f_i(x_2))^2}$ denoting the Euclidean distance between x_1 and x_2 in objective space. Note that we here assume that the objective space is of a reasonable dimensionality. For high-dimensional real spaces, the Euclidean distance is not a good distance measure anymore. See Houle *et al.* (2010) for more information and a rank-based solution to this problem.

Consider now that we are given a set of such solutions $X^* \subset X$. We do not make any assumptions about this set and about how this set has been generated; for example, it can be the output of a multiobjective optimizer, and it can contain both dominated and non-dominated solutions. This set may be difficult to interpret, especially if there are many solutions, many objectives and if the solutions have a complex decision space representation. We therefore would like to generate a partitioning of this set; that is, we would like to group the solutions into clusters to ease the interpretation of the optimization results.

Definition 1
A cluster $c \subseteq X^*$ is a subset of all solutions in the given set X^*.

Definition 2
A partitioning $C = \{c_1, \ldots, c_k\}$ is a set of k clusters such that each solution is included in exactly one cluster, that is, $\forall x \in X^* : (\exists c_i \in C : x \in c_i)$, and no

solutions is included in more than one cluster, that is, $x_i \in c_j \wedge x_i \in c_l \Rightarrow j = l$.

But what is a good partitioning? Usually, a good partitioning is one where the clusters are compact and well separated. This means that solutions within a cluster should be close to each other (i.e. the cluster has a small intracluster distance), and solutions of different clusters should be far from each other (i.e. the clusters have a large intercluster distance). In the literature (Xu and Wunsch, 2009), these two measures are usually combined into one goodness measure, the so-called validity index. We therefore assume that we are given such a validity index $V : (D, d) \rightarrow \mathbb{R}$. Here, D is an arbitrary space, and d is a distance measure defined on D, that is, $d : D^2 \rightarrow \mathbb{R}$. Note that many validity indices cannot cope with partitionings that contain only one cluster (which in turn contains all solutions). Also, they sometimes have problems with clusters that only contain one solution. We therefore assume that a feasible partitioning must contain at least two clusters, and each cluster must contain at least two solutions. This reduces the possible number of clusters k to the interval $k \in \left[2, \left\lfloor \frac{|X^*|}{2} \right\rfloor\right]$.

In this paper, we would like to find a partitioning that is good both in decision space and in objective space. We therefore have two objectives. The first is the validity index in objective space, and the second is the validity index in decision space. Whether these two indices are conflicting or not depends on the given solution set X^*. There may well be solution sets where a good partitioning in objective space does not result in a good decision space partitioning and vice versa, for example, for optimization problems where radically different designs can lead to similar objective space values. Depending on the chosen distance measure, it can also happen that two solutions with a low distance to each other still do have quite dissimilar objective values, in particular if the distance measure does not capture all differences between the solutions.

As two conflicting objectives generally lead to a tradeoff front, we here suggest to optimize the two validity indices as a biobjective problem in order to find that front. This has the advantage that the two indices do not have to be combined into one goodness measure a priori. Furthermore, the tradeoff between a good partitioning in decision space and in objective space can be visualized, and the user can then choose one of the partitionings depending on which space is more important to him or her. The optimization problem can therefore be stated as follows:

Find a partitioning C^* such that $V(f(C^*), d_O)$ and $V(C^*, d_D)$ are optimal. Here, $V(f(C^*), d_O)$ is the validity index calculated on the objective space values of the solutions in C^*, and $V(C^*, d_D)$ is the validity index calculated on the decision space values. d_O and d_D are the distance measures in objective and decision spaces, as defined in the first two paragraphs of this section.

3. RELATED WORK

This problem is closely related to traditional clustering, which aims at finding groups of points in such a way that the points within a cluster are as similar as possible, whereas points belonging to different clusters should be well distinguishable. Note that clustering is an unsupervised process that groups solutions on the basis of how near they are to each other. This differs from classification, which uses supervised learning to derive rules to assign solutions to groups by using training data, that is, given assignments that are known to be correct. Clustering problems have been known for a long time (see, e.g. the book of Xu and Wunsch, 2009, for a good overview of the field and an introduction into standard clustering techniques, including *partitional clustering*, which will be used in this work). Other techniques that will not be considered in this paper because they either place some assumptions on the solution space or do not produce crisp clusters are hierarchical, neural network based, kernel based, sequential or fuzzy clustering techniques.

The clustering problem we would like to solve in this paper differs from traditional clustering as the considered points are characterized by two aspects, namely the decision space representation of solutions as well as their objective space values. We would like to group solutions such that the clusters are close in objective space but at the same time exhibit strong similarities in decision space. Note that this is not the same as multiobjective clustering, as it is, for example, described in Handl and Knowles (2007). Multiobjective clustering aims at solving common problems in standard clustering, such as setting the tradeoff between cluster compactness, cluster separation and cluster number. It does so by transforming the clustering problem into a biobjective problem, a process that is also known as multiobjectivization, where the first goal is to optimize the cluster compactness and the second goal is to optimize cluster separation.

Clustering of data, which is characterized by more than one aspect, has recently gained attention in the field of bioinformatics, where for instance genes need to be grouped according to their mRNA expression profiles and their protein interaction partners. A commonly used approach combines this data into one matrix and then applies conventional clustering techniques (Eisen *et al.*, 1998). In our case, the cluster measures are different for the objective space and the decision space, so merging the two spaces is not an option. Other approaches consider both datasets separately but are designed to find only a single best cluster (Calonder *et al.*, 2006; Kutalik *et al.*, 2008). In this study, however, we would like to find multiple groups of solutions. Bushel *et al.* (2007) used a common distance measure, that is, the sum of Euclidean distances in both spaces, and then applied the *k*-means clustering algorithm to derive the groups. In our problem however, we are considering datasets where the best partitioning in decision space might be very different from the best partitioning in objective space, and we would like to generate the tradeoff solutions in between. Pollard and van der Laan (2002) applied iterative clustering, which means that the data are first clustered in one space, and the resulting clusters are then clustered again in the other space. This process can be repeated, or the order of the spaces can be reversed. This approach is very similar to the approach proposed by Aittokoski *et al.* (2009), who applied a modified *k*-means algorithm to cluster the solutions in objective space. For a refinement, the same algorithm can be applied to group the solutions of individual clusters in decision space. Finally, Narayanan *et al.* (2010) proposed a measure to quantify the goodness of clusters in different spaces. It assumes that each space can be transformed into a graph, where the nodes are the genes and the edges are the relations between genes. Here, different relations can be modelled in different graphs. The measure then calculates for each cluster a score on each graph, and the worst score over all graphs is selected as the representative score for that cluster. The partitioning goodness measure then is defined as the sum of these representative scores of all clusters.

Some recent efforts have been undertaken in order to infer relationships between decision and objective spaces, which helps to extract design principles that can be useful to the decision maker. One such method is called ‘innovization’ (innovation through optimization) (Deb and Srinivasan, 2006). To be able to apply innovization, it is assumed that the decision space is built from real and/or discrete decision variables that can take certain values. In earlier innovization approaches (Deb and Srinivasan, 2006), solutions were examined manually on a specific problem to derive interesting facts about variables such as common variable settings, variable importance, and relations between variable settings and objective values. More recent approaches (Bandaru and Deb, 2011) automate this process by first using clustering in objective space and then fitting some basis functions to model the data in the cluster.

Other approaches aim at visualizing the Pareto-front and/or the Pareto-optimal solutions in decision space and inferring design principles from these visualizations. One such approach is using self-organizing maps (Obayashi and Sasaki, 2003), where high-dimensional decision and objective spaces are mapped to two-dimensional maps. Another approach is using heat maps (Pryke *et al.*, 2006), where real-valued variable and/or objective vectors of a set of solutions are plotted as coloured heat maps. Both approaches assume that the decision space is a real-valued space.

Some work has also been carried out to do feature extraction. Sheng *et al.* (2008) assumed that each solution can be described as a set of features, which can, but do not have to be equal to the decision variables. They then optimized a partitioning using an evolutionary algorithm, where they also evolved a subset of features that is to be taken into account when calculating the partitioning goodness. Sugimura *et al.* (2009) also assumed that there are design variables and mines for design rules that specify which variable settings lead to which fitness levels. Preliminary work of the author (Ulrich *et al.*, 2008) applies biclustering to the (binary) decision space to identify characteristic subsets of decision variables, called modules. These modules are then used to cluster the solutions. This approach has two major shortcomings. First, it focuses on the decision space, and a straightforward integration of objective space information is only possible if there are no more than two objectives. Second, the used biclustering method only works for binary decision spaces. Note that all of the previously mentioned approaches make some assumptions on the decision space, that is, that there is a given set of design variables. Considering the bridge problem described in the introduction, it might be difficult to define the space of all possible bridges with the use of design variables. Our approach aims at such problems with complex decision spaces, as the only requirement of our approach is that it is possible to measure the distance between any two solutions.

There has been a multitude of approaches to do clustering using evolutionary algorithms (see, e.g. Hruschka *et al.*, 2009, for a comprehensive overview of current approaches). These approaches mainly differ in the used representations, variation operators, fitness functions (i.e. the used cluster validity index) and whether the number of clusters is variable or is assumed to be fixed. Bandyopadhyay and Maulik (2001) also did a comparison of three cluster validity indices.

Also, clustering has been used to prune a given set of tradeoff solutions, for example, produced by a multi-objective optimizer in order to help the decision maker. Typically, this clustering is carried out solely in objective space. Taboada and Coit (2007) applied the *k*-means algorithm for all possible number of clusters. Morse (1980) used both partitional and hierarchical clustering. Rosenman and Gero (1985) tackled the problem of differently scaled objectives.

Finally, there has been some work that aims at maintaining diversity in decision space during optimization (e.g. Rudolph *et al*., 2007; Ulrich *et al*., 2010). If there are so-called preimages (i.e. distinct regions) in the decision space that map to the whole Pareto-optimal front, a decision maker might be interested in finding all of those preimages. In such cases, clustering the solutions not only in objective space but also in decision space is advantageous.

4. PAN: THE PARETO-FRONT ANALYSER

Clustering problems in general are hard to solve. A simultaneous clustering in two spaces is even more challenging, and it is not clear how an algorithm should be designed to achieve good clusters, especially if several cluster validity indices are considered. In this paper, we therefore propose PAN, an evolutionary algorithm, to optimize the biobjective problem defined in Section 2. The general framework of our evolutionary algorithm is shown in Algorithm 1. This is a standard form of an evolutionary algorithm, where variation and selection is iteratively applied for a fixed number of iterations. Note that in PAN, each solution corresponds to a partitioning. The population P therefore is a set of partitionings, and the objective functions are the partitioning goodness measures in decision space and in objective space.

Algorithm 1 General framework of an evolutionary algorithm. Input parameters: population size n; minimization is carried out for g generations.

```
function EA($n$, $g$)
    Initialize population $P$ randomly with n partitionings
    for $g$ generations do
        $P'$ = VARIATE ($P$,$n$)/* generate n offspring */
        $P$ = SELECT ($P \cup P'$, $n$)/* select n partitionings */
    return $P$
end function
```

For the selection procedure, we opted to go for the standard greedy hypervolume-based selection, which is shown in Algorithm 2, where HYP (P) is the hypervolume of population P. The hypervolume in turn is calculated on the objective values of the solutions, which is defined by the selected cluster validity index.

Algorithm 2 Selection procedure. Input parameters: population P, number of partitionings to select n.

```
function SELECT ($P$, $n$)
    while $|P| > n$ do
        /* remove partitioning with smallest contribution */
        $P = P \setminus \{\arg\min_{pi \in P} (\mathrm{HYP}(P) - \mathrm{HYP}(P \setminus pi))\}$
    return $P$
end function
```

The variation procedure is shown in Algorithm 3. We here assume that the number of offspring to generate is equal to the population size. Also, we are using random sampling without replacement as a mating selection scheme. Note that there are some constraints on the partitionings, namely that each partitioning must at least contain two clusters and that each cluster must at least contain two solutions. The functions ISVALID and ISINVALID check whether a given partitioning is valid or invalid. We here deal with these constraints by using a repeat strategy; that is, for each parent pair selected during mating selection, we keep generating offspring until two feasible offspring have been found. The recombination and mutation of course depends on the selected representation and will be described in more detail in Section 4.2.

Algorithm 3 Variation procedure. Input parameters: population P, (even) number of offspring n; recombination probability p_R.

```
function VARIATE ($P, n, p_R$)
    for 1 to $n/2$ do
        set $o_1$ and $o_2$ to an invalid partitioning
        while ISINVALID($o_1$) or ISINVALID($o_2$) do
            /* randomly select two parents from P */
            $\{p_1, p_2\}$ =MATINGSELECTION($P$)
            $o'_1 = p_1$, $o'_2 = p_2$ /* set offspring to parents */
            With probability $p_R$: $\{o'_1, o'_2$ =RECOMBINE($p_1, p_2$)
            $o'_1$ =MUTATE($o'_1$)
            $o'_2$ =MUTATE($o'_2$)
            if ISINVALID($o_1$) and ISVALID($o'_1$) then
                $o_1 = o'_1$
            if ISINVALID($o_2$) and ISVALID($o'_2$) then
                $o_2 = o'_2$
        $P = P \cup \{o_1, o_2\}$
    return $P'$
end function
```

4.1. Speed up by local heuristic

Preliminary tests (see also Section 5.1) showed that without any speedup, PAN with an arbitrary representation and validity index takes a long time to reach satisfying partitionings. To speed up the search, we therefore propose to integrate a local heuristic into the search. One of the most common clustering algorithms is the k-means algorithm (MacQueen, 1967). The k-means algorithm is known to converge fast towards the nearest local optimum, which makes it well suitable as a local heuristic during optimization.

To integrate the local heuristic, we propose to locally optimize the offspring partitionings both in decision space and in objective space and then select the future parents from the set containing both original offspring, offspring locally optimized for partitioning goodness in objective space and offspring locally optimized for partitioning goodness in decision space. The adapted general framework of PAN that incorporates the local search is shown in Algorithm 4.

Algorithm 4 PAN algorithm with local search. Input parameters: population size n; minimization is carried out for g generations.

```
function PAN($n$, $g$)
    Initialize population $P$ randomly with n solutions
    for $g$ generations do
        $P'$ = VARIATE($P$, $n$)/* generate n offspring */
        /* apply local optimization in both spaces */
        $P'_o$ = LOCALOPT($P'$, obj)
        $P'_d$ = LOCALOPT($P'$, dec)
        $P$ = SELECT($P \cup P' \cup P'_o \cup P'_d$, $n$)/* select n solutions */
    Return $P$
end function
```

Note that the original k-means algorithm makes use of the cluster centroids, which assumes that the solutions are given in Euclidean space. As we only require pairwise distances in decision space, we therefore use the k-medoids (Kaufman and Rousseeuw, 1990) algorithm instead, which is an adapted version of k-means that works with cluster medoids instead of centroids (see also Section 4.3 for more details about cluster medoids).

4.2. Representation

When designing an evolutionary algorithm, a suitable representation has to be chosen for the problem at hand in order to code the different solutions, in this case partitionings. In the literature, several representations are used for clustering problems, namely the centroid, graph, integer and direct representations (e.g. Hruschka *et al*., 2009).

4.2.1. Centroid representation. The centroid representation is used by several authors (Das *et al*., 2009; Kundu *et al*., 2009) and codes only the cluster centroids. Each solution is then assigned to the nearest centroid. This is similar to the cluster allocation of the well-known k-means clustering algorithm (MacQueen, 1967). The centroid representation has the advantage that it considerably reduces the search space, as only a small number of centroids have to be chosen. The disadvantage is that first, this representation assumes that the solutions are given in Euclidean space, and second, it is not at all clear how one centroid can be decoded into two spaces. We therefore need a representation that directly represents the solutions assignment to clusters, without making any assumptions about the used spaces.

4.2.2. Graph representation. Park and Song (1998) suggested the graph representation, which is an adjacency list of length n, where n is the number of solutions. The ith value in the list codes one link that says to which other solution the ith solution is connected to. The connections of the whole adjacency list return a graph, where the clusters are the unconnected subgraphs. Handl and Knowles (2005b) did extensive tests with this representation and found that it works satisfactorily. The advantage of this representation is that standard variation operators can be applied. Here, we follow Handl and Knowles and use uniform crossover with switching probability of 0.5 for each element to do recombination and randomly change one element in each mutation. Note that Handl and Knowles proposed to reduce the search space by allowing each solution to be only connected to its L nearest neighbours. Also, Handl and Knowles state that links to further away individuals are less favourable than links to close neighbours and should therefore be mutated with a higher probability (Handl and Knowles, 2005a). To keep the comparison between different representations fair, we do not make use of these techniques.

When using the graph representation, applying the local heuristic using k-medoids is not straightforward as the locally optimized partitioning p_i^{opt} might look quite different from the original partitioning p_i. If so, it is not quite clear how to incorporate these changes into the original graph structure while keeping as many common links as possible. We here use the following approach. First, starting from the original partitioning p_i, all links between solutions that are not in the same cluster in the optimized partitioning

p_i^{opt} are removed. Then, for each remaining cluster in p_i, an unweighted minimum spanning tree is calculated, and all links not present in the minimum spanning tree are removed. Then, all links that have been removed in the previous two steps are reinserted in a random manner, and it is checked whether the new partitioning p_i corresponds to the optimized one p_i^{opt}. If not, another random assignment is selected. If no assignment is found that produces the optimized partitioning, the locally optimized partitioning is discarded.

4.2.3. Integer representation. Another representation we consider in this paper is called the integer representation. It is coded by an integer string $x \in \left\{1, 2, \ldots, \left\lfloor \frac{n}{2} \right\rfloor \right\}^n$ of length n, where n is the number of solutions. Solutions with the same integer value are assigned to the same cluster. As a mutation operator, we use single-point mutation, where one randomly chosen position in the string is assigned a randomly chosen new integer value already present in the string, that is, $\upsilon_{\text{new}} \in \{x\}$. As a recombination operator, we use uniform crossover, where for each position in both parent strings, the two integers are exchanged with probability 0.5.

4.2.4. Direct representation. We also suggest to use a direct representation, inspired by the work of Falkenauer (1998). The direct representation stores a (variable length) list of clusters, where each cluster in turn is a list of solutions. We then define three mutation operators for this representation:

- Move operator: moves a randomly selected solution to a randomly selected other cluster, with probability p_m;
- Merge operator: merges two randomly selected clusters, with probability p_u; and
- Split operator: splits a randomly selected cluster into two random parts, with probability p_s.

As a recombination operator, we suggest to use an operator proposed by Falkenauer (1998). It resembles a two-point crossover in the following way: given two parents p_1 and p_2, from which we want to create two offspring o_1 and o_2. First, we set $o_1 = p_2$ and $o_2 = p_2$. Then, we choose two random cut points in the cluster list of both parents. The clusters between the two cut points of p_1 are added to o_2 in the position after the first cut point in p_2. Now there are several original clusters in o_2 that contain the same solutions as the clusters added from p_1. Therefore, these duplicate solutions are removed from their clusters. The second offspring is generated in the same way, with the roles of the parents reversed.

4.3. Validity indices

As stated in Section 2, we would like to find partitionings that have a good cluster validity index both in objective space and in decision space. In the literature, a multitude of different validity indices can be found (e.g. Bandyopadhyay and Maulik, 2001; Halkidi *et al*., 2002; Hruschka *et al*., 2009; Xu and Wunsch, 2009). They usually combine the two clustering goals, that is, cluster compactness and cluster separation, into one goodness measure. Usually, these validity indices are used to find the correct number of clusters to a given clustering problem. To do so, clustering optimizers that take the number of clusters k as a parameter (e.g. the well-known k-means algorithm) are run for different values of k, and the resulting partitioning that achieves the highest cluster validity index is chosen to be the correct one. Optimizing such a validity index will therefore automatically lead to a partitioning with the correct number of clusters (see also Milligan and Cooper, 1985; Bandyopadhyay and Maulik, 2001; Halkidi *et al*., 2002, for overviews over indices that are used to identify the correct number of clusters).

Many of the validity indices found in the literature assume that the points to be clustered are given in Euclidean space. Most of the time, they assume that a cluster centroid can be calculated, where in each dimension, the centroid value is the mean value of all solutions in the cluster and in the respective dimension. Examples for such indices are the Davies–Bouldin index (Davies and Bouldin, 1979), the CS index (Chou *et al*., 2004), some variants of the Dunn index (Bezdek and Pal, 1995), the SD index (Halkidi *et al*., 2000), the $I(k)$ index (Bandyopadhyay and Maulik, 2001) and the adapted silhouette index (Hruschka *et al*., 2006). As we only assume that we are given pairwise distances but without any information about the underlying decision variables, the cluster centroids cannot be calculated. To solve that problem, we here propose to use the medoids instead of the centroids. The medoid of a cluster is the solution with the smallest average distance to all other solutions in the cluster (Kaufman and Rousseeuw, 1987). Note that, whereas the calculation of the centroid is linear in the number of solutions, the calculation of the medoid is quadratic. See Handl (2005) for a sampling approach that faces this issue and speeds up the medoid calculation.

Also, there are some validity indices that do not only use centroids but also use the notion of a direction in the solution space, for example, the S_Dbw index (Halkidi and Vazirgiannis, 2001) or the ReD index (Jornsten *et al*., 2002). Such indices cannot be used for our problem.

Finally, we decided to test the following nine indices. The silhouettes index (Rousseeuw, 1987) $S(C)$ with an adaptation for minimization $S(C) = -(S_{\text{orig}}(C) - 1)$, where $S_{\text{orig}}(C)$ is the silhouettes index as defined in the original paper, the adapted silhouettes index (Hruschka *et al*., 2006) $AS(C) = -(AS_{\text{orig}}(C) - 1)$, the Dunn index (Dunn, 1974) $D(C) = -D_{\text{orig}}(C)$, the generalized Dunn index (Bezdek and Pal, 1995) $GD(C) = -GD_{\text{orig}}(C)$, the VRC index (Calinski and Harabasz, 1974) $VRC(C) = -VRC_{\text{orig}}(C)$, the Davies–Bouldin index (Davies and Bouldin, 1979) $DB(C) = DB_{\text{orig}}(C)$, the CS index (Chou *et al*., 2004) $CS(C) = CS_{\text{orig}}(C)$, the I index (Bandyopadhyay and Maulik, 2001) $I(C) = -I_{\text{orig}}(C)$ and the SD index (Halkidi *et al*., 2000) $SD(C) = SD_{\text{orig}}(C)$.

4.4. Practical considerations

The PAN algorithm only makes a few assumptions about the dataset at hand. The first one is that the best partitioning in objective space is different from the best partitioning in decision space. If the two clustering goals are not conflicting, there is no set of tradeoff partitionings but a single best partitioning. In this case, the final PAN population will contain a partitioning that dominates all others.

The second assumption is that each cluster contains at least two solutions, because some validity indices cannot handle clusters with only one solution. We therefore suggest to do a data cleaning step where outliers, that is, solutions that have a large distance to all other solutions, both in objective space and in decision space, are identified by hand and removed prior to clustering.

5. SELECTION OF VALIDITY INDEX AND REPRESENTATION

Clustering problems in general are hard to solve, and the search space is huge, even for a reasonable number of points to be clustered. If the optimization should work satisfactorily, the used representation and partitioning goodness measure have to be selected carefully. This is because some indices might introduce plateaus or many local optima. In this section, we try to find a combination of validity index and representation that performs satisfactorily on a clustering problem.

Usually, to test which validity index/representation combination works best on a standard clustering problem, the different combinations are tested on datasets where the optimal partitionings are known. Then, the combination whose result is closest to the known partitioning is chosen as the best one. In this paper, however, the dataset is given in two spaces, namely the objective space and the decision space, and a good partitioning should be good in both spaces. Assuming that the best partitioning in one space is not equal to the best partitioning in the second space, we are given the choice between different tradeoff partitionings. Unfortunately, it is not clear what qualifies as a good tradeoff partitioning. Moreover, a tradeoff that is good with respect to one validity index can be quite bad with respect to another validity index. However, we know that all combinations should be able to find those two partitionings that are best in either the first or the second space, because these two partitionings are Pareto optimal, independently of the chosen validity index (assuming that the validity index is actually best for the known optimal partitioning).

Therefore, we test our combinations by constructing different clustering problems where we know the optimal partitionings in both spaces in advance and see whether the combinations can find the two extremal partitionings (those best in one of the two spaces) in the same run. We here selected three test cases. The first test case is the simplest where both spaces to be clustered contain four clearly distinguishable clusters with five solutions each. The second test case has clusters with different numbers of solutions to test PAN's capability to recognize differently sized clusters. Finally, the third test case has a larger set of solutions to be clustered in order to test PAN's capability to achieve good partitionings even for a large number of solutions.

We use the same experimental setup for all three test cases; that is, we use a population size of 10 for 500 generations, Euclidean distance as a distance measure (where we normalize all pairwise distances to lie in the interval $[0, 1]$). For each setting, we do 30 runs, with recombination probability $p_R = 0.7$ and $p_m = 0.6$, $p_u = 0.2, p_s = 0.2$ for the mutation operator of the direct representation.

To compare the results, we consider two aspects. First, we check whether the partitionings that PAN finds to be optimal either in decision or objective space correspond to the expected optimal partitionings. And second, we measure the minimum number of function evaluations that is needed to find the optimal partitionings in both spaces.

5.1. First test case: proof of concept and validation of local heuristic

The simplest test case is shown in Figure 1, which shows the optimal partitioning in both spaces. In both spaces, the best partitioning consists of four clusters of five solutions each. However, these best partitionings

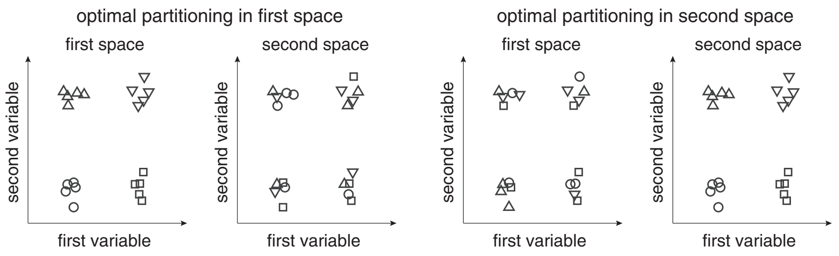

Figure 1. Points to be clustered for the first test case. The left two plots show the optimal partitioning in the first space, and the right two plots show the optimal partitioning in the second space. Both pairs of plots show the points in the first/second space in their first/second plot.

do not correspond to each other, as can be seen in the figure, where the left two plots show the best partitioning in the first space (and the corresponding partitioning in the second space), whereas the right two plots show the best partitioning in the second space (and the corresponding partitioning in the first space). For reasons of simplicity, we used the same location of points in both spaces. Note that if using the Pareto-optimal set of an optimization problem, the first space might be the objective space and the second space might be the decision space. We applied PAN with and without the local heuristic to get a feeling about the speedup when adding the local heuristic.

In both cases, we observed that the SD index found suboptimal partitionings to have a better validity index value than the known optimal ones. When inspecting these partitionings, it can be seen that the reason for this behaviour is the use of the medoid instead of the centroid. If one cluster contains solutions from all four optimal clusters, the centroid will lie in the centre of the solutions, whereas the medoid has to be one of the actual solutions and therefore is quite far from the centroid, which in turn causes problems when calculating the SD index. The SD index therefore cannot be used for the optimization.

For the remaining indices, the number of function evaluations after which both optima have been found without the local heuristic is shown in the left plot of Figure 2. It can be seen that no combination reaches both optima within 5000 function evaluations in all runs, which is an indication for a low convergence speed. Nevertheless, VRC, S and AS indices with direct representation as well as the DB index with integer representation seem to work better than the remaining combinations.

To tackle the slow convergence speed, we now add the local heuristic. The number of function evaluations for reaching the known optima is shown in the right plot of Figure 2. As can be seen from the figure, all combinations (with the exception of the GD and DB indices with integer representation that did not reach both optima in 5000 function evaluations in one out of all 30 runs) reach both optima within 800 function evaluations. Moreover, it can be seen that the direct and integer representations find the optima faster than the graph representation.

5.2. Second test case: irregular clusters

In the last section, the known optimal clusters were all of the same size, and both spaces had the same

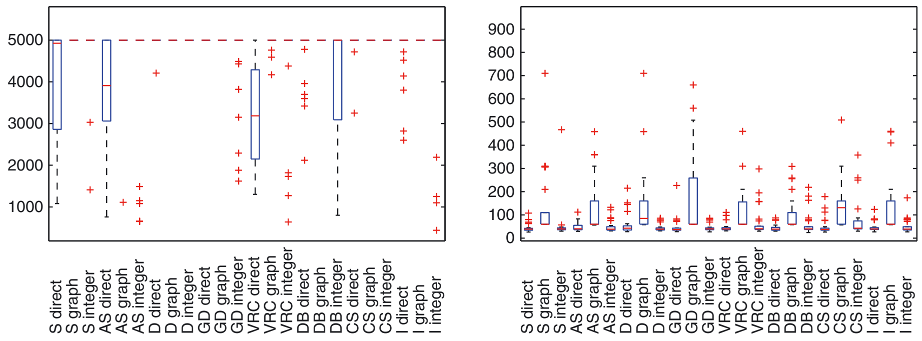

Figure 2. Number of function evaluations (smaller is better) after which both optima have been found for different validity index/representation pairs, without local heuristic (left plot) and with local heuristic (right plot).

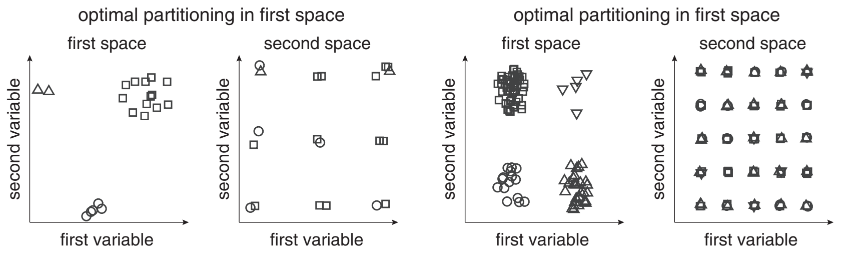

Figure 3. Points to be clustered for the second (left two plots) and third (right two plots) test cases. Both pairs of plots show the optimal partitioning in the first space (with the corresponding partitioning in the second space in the second plot of the respective plot pair).

optimal number of clusters (i.e. four). The goal of this section is to see how (i) PAN performs if the best clusters are of different sizes and (ii) whether PAN struggles with cases where the optimal number of clusters is quite different in the two spaces. The corresponding problem is shown in Figure 3. Note that, in the first space, there are three clusters with 2, 5 and 13 solutions each, whereas in the second space, there is a more regular structure with eight clusters of two solutions each and one cluster with four solutions.

When looking at the results, it has been found that the VRC index, the I index as well as the SD index all find suboptimal partitionings that have a better value than the known optimal ones and therefore cannot be used for the optimization. For the remaining validity indices, the number of function evaluations after which both optima have been found is shown in the left plot of Figure 4. It can be seen that the direct representation is faster for the S, D and AS indices, and not worse in the other indices than the graph and integer representations.

5.3. Third test case: larger dataset

Considering the results from the previous two test cases, it was found that the direct representation works better than the other two representations. Also, the VRC, I and SD indices cannot be used, because their optimal partitionings are known to be suboptimal. The remaining indices seem to perform satisfactorily, so we will test these indices with direct representation on a larger dataset. The dataset under consideration is shown in Figure 3. In the first space, there are four distinct clusters with 5, 15, 30 and 50 solutions each. In the second space, there are 25 evenly spaced clusters with four solutions each.

The results are shown in the right plot of Figure 4. It can be seen that PAN both with the D and GD indices do not find both optima within 5000 function evaluations. Also, the S and CS indices perform slightly better than the AS and DB indices. Finally, it has been found that the S index takes much longer to compute than the remaining indices, as it needs to calculate the pairwise distances between all solutions.

5.4. Test case summary

In this section, we considered several artificial datasets in order to find a good configuration for PAN. First, it could be observed that the speedup proposed in Section 1 decreases the required number of function evaluations to reach the known optimal partitionings significantly. Second, it was found that the SD, VRC and I indices sometimes find suboptimal partitionings that have a better partitioning goodness value than the known optimal partitionings. Also, the D and GD indices do not reach the optimal partitioning in a reasonable number of function evaluations for larger datasets. And finally, the direct representation has a better performance than the graph and integer representations, especially on larger datasets. In conclusion, it was found that a good combination for PAN is to use direct representation with the S, AS, DB or CS index.

6. RESULTS

This section first compares the proposed algorithm with the standard approach of iteratively applying the *k*-medoids algorithm. Then, the method is applied first to a knapsack problem and then to a real-world bridge construction problem, and its results are qualitatively inspected.

6.1. Comparison with *k*-medoids

The goal of this section is to validate the multiobjective approach. To this end, we compare the achieved hypervolume with the hypervolume obtained by the standard method of applying *k*-medoids iteratively. In more detail, this is performed in the following way. We apply *k*-medoids several times for all possible cluster numbers. Each time, we cluster the solutions twice, once in

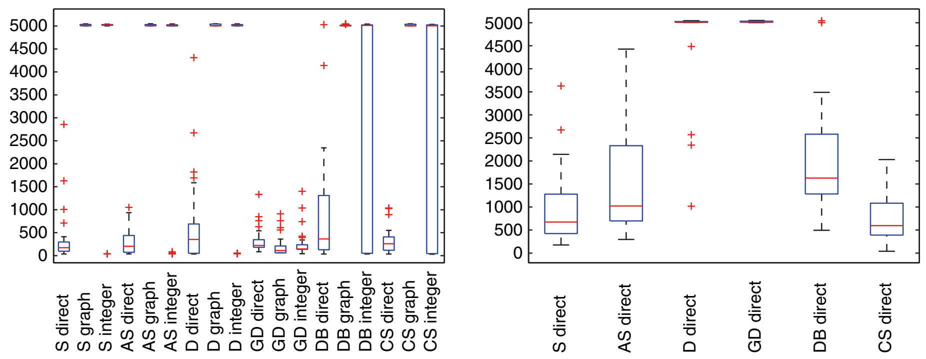

Figure 4. Number of function evaluations (smaller is better) after which both optima have been found for all representation/validity index pairs, with local heuristic.

decision space and once in objective space, and check whether the optimized partitionings satisfy the constraints (i.e. they contain at least two clusters, where each cluster must at least contain two solutions). For all partitionings that satisfy the constraints, we calculate the cluster goodness values in decision space and in objective space. Finally, to compare the resulting population with PAN, we reduce the number of achieved solutions to the population size used with PAN, using PAN's selection procedure.

We compared k-medoids with PAN on the third test case, where PAN uses direct representation and the same indices tested in Section 3. PAN's population size again is 10, and it is run for 5000 function evaluations. The iterative k-medoids algorithm, on the other hand, is applied for all cluster numbers in the interval $[2, 50]$, with $\lceil 5000/49 \rceil = 103$ restarts for each cluster number. This way, both PAN and the iterative k-medoids algorithm use the same number of calls to the actual k-medoid algorithm.

The corresponding hypervolume values for the different validity indices are shown in Table I. According to a Kruskal–Wallis test performed on the data as described by Conover (1999), with the Conover–Inman procedure, Fisher's least significant difference method performed on ranks and a significance level of 1%, PAN is always significantly better than the k-medoids algorithm, except for the D index, which has many outliers. This indicates that some partitionings found by PAN cannot be achieved by using k-medoids. Instead, slight variations of partitionings produced by k-medoids might have a high gain in one space but at the same time not much loss in the other space.

Table I. Mean and standard variation of achieved hypervolume values of PAN and the iterated k-medoids algorithm for six indices

Index	PAN	k-medoids
S	0.68 ± 0.27	0.36 ± 0.19
AS	0.80 ± 0.23	0.22 ± 0.13
D	0.20 ± 0.32	0.10 ± 0.02
GD	0.52 ± 0.29	0.12 ± 0.04
DB	0.63 ± 0.25	0.29 ± 0.11
CS	0.75 ± 0.18	0.34 ± 0.17

For each index, all achieved hypervolume values were normalized such that the minimum/maximum hypervolume have the values 0/1.

6.2. Application to knapsack problem

First, we applied PAN to a simple biobjective knapsack problem. Here, we consider a problem with 150 items where each item i has two randomly chosen profits p_i^1 and p_i^2 and weights w_i^1 and w_i^2, where p_i^1, p_i^2, w_i^1 and w_i^2 are chosen uniformly and at random in the interval $[10, 100]$. The problem can therefore be viewed as a selection problem, where a subset of the 150 items has to be selected, which will be evaluated in two separate knapsacks, where each item has a different profit and weight in each knapsack. Each solution $x = \{x_1, x_2, \ldots, x_{150}\} \in \{0, 1\}^{150}$ is a binary string of length 150, saying for each item whether it is selected or not. The biobjective knapsack problem is a constrained problem, where for each feasible solution x, $\sum_{i=1}^{150} x_i \cdot w_i^1 \leq 0.2 \cdot \sum_{i=1}^{150} w_i^1$ and $\sum_{i=1}^{150} x_i \cdot w_i^2 \leq 0.2 \cdot \sum_{i=1}^{150} w_i^2$ must hold; that is, the total weight of all selected items in each knapsack must not exceed 20% of the total weight of all items of that knapsack. The objectives then are the sum of profits of each knapsack, that is, the first objective is to maximize $\sum_{i=1}^{150} x_i \cdot p_i^1$, and the second objective is to maximize $\sum_{i=1}^{150} x_i \cdot p_i^2$. These objectives can be transformed easily into minimization problems using the following formula:

$$f_1(x) = \sum_{i=1}^{150} p_i^1 - \sum_{i=1}^{150} xi \cdot p_i^1$$
$$f_2(x) = \sum_{i=1}^{150} p_i^2 - \sum_{i=1}^{150} xi \cdot p_i^2$$

We used the integer programming problem solver CPLEX (IBM, USA) to generate the exact Pareto-optimal front for one instance of this knapsack problem with 150 items. The resulting front contains 138 Pareto-optimal solutions, which we will now cluster using PAN. We applied PAN using the AS, CS and DB indices, using direct representation, a population size of 20 and running PAN for 100 000 function evaluations. We found that the AS index has a tendency to produce many small clusters, whereas the CS produces two small clusters and one large cluster. The DB index, instead, produces a few clusters of reasonable size. We will only show the results of the DB index in the following. When looking more closely at the found partitionings shown in Figure 5, it was found that they can be classified in two templates, one that contains three clusters and one that contains two clusters. All partitionings are very similar to one of these two partitioning templates. We will look at a partitioning with three clusters in the following.

To visualize one solution in decision space, the profits of the chosen and discarded items can be plotted. Note that there are 17 items that are selected in all Pareto-optimal solutions and 75 items that are never selected in any of the Pareto-optimal solutions. The profits of these items will not be plotted, although an engineer might certainly look at them to learn more about the problem at hand. To interpret differences and similarities of clusters, only the 58 items that are selected in some solutions but not in others are plotted. To plot a whole cluster, the cluster medoid and the solution furthest from the medoid are calculated, and the items that are selected/not selected in those two representative solutions are plotted. Also, it is indicated which items are selected in all/no solutions of that cluster.

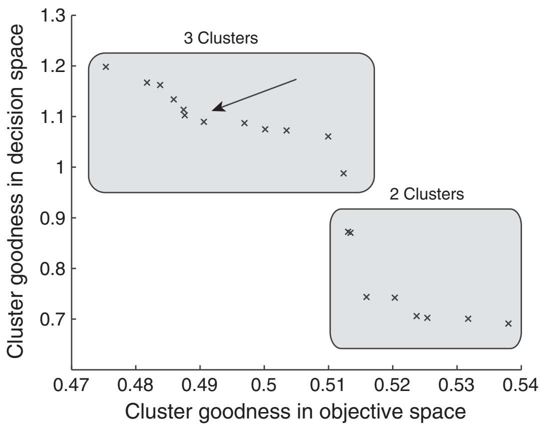

Figure 5. Partitionings resulting from one PAN run on the knapsack problem using the Davies–Bouldin index. All partitionings either had two or three clusters (as indicated). The chosen partitioning, which will be inspected in more detail, is indicated with an arrow.

The results are shown in Figure 6. As can be seen, there are two large clusters and one small cluster, where the two large clusters cover the two extremal regions of the Pareto front and the small cluster covers the middle region. When looking at the decision space, the connection between selected items and location of the solution on the front can be seen. Cluster 1 contains the solutions with the highest profit in the first knapsack and at the same time with the lowest profit in the second knapsack (remember that the profits in the left plot of Figure 6 are transformed to yield a minimization problem). Several items are selected/not selected in all solutions of this first cluster, and the selected solutions all have a good profit in the first knapsack. In the third cluster, the opposite holds, namely the items selected in all solutions mainly have a good profit in the second knapsack, leading to solutions with a good overall profit in the second knapsack. Finally, the middle cluster contains solutions that selected items from the whole range of profits in both knapsacks. As for the difference between cluster medoids and solutions furthest from the medoid, it can be noted that the Hamming distance between the medoid and furthest solution are 12, 8 and 12 items for the first, second and third clusters, respectively. For the whole dataset, the Hamming distance between solutions varies between 2 and 43 items (remember that from the 150 items, only 58 are not selected/deselected in all solutions), with a mean distance of 15.22 ± 7.78.

6.3. Application to bridge construction problem

We also applied our algorithm to a real-world problem. As a problem, we selected the bridge construction problem that is inspired by Bader (2010), where the goal is to build a truss bridge that can carry a fixed load. An example bridge can be seen in Figure 7. Each bridge basically is a set of nodes, with connections between certain node pairs. All bridges have to be built in the following framework. First, there are two fixed nodes, shown as black circles in the figure, to which the bridge is connected. Note that in accordance with standard truss analysis, the fixed node on the left side of the bridge is fixed both in horizontal and vertical directions, whereas the fixed node on the right side of the bridge is fixed only in the vertical direction. Each bridge has a set of six horizontal connections, called the decks, over

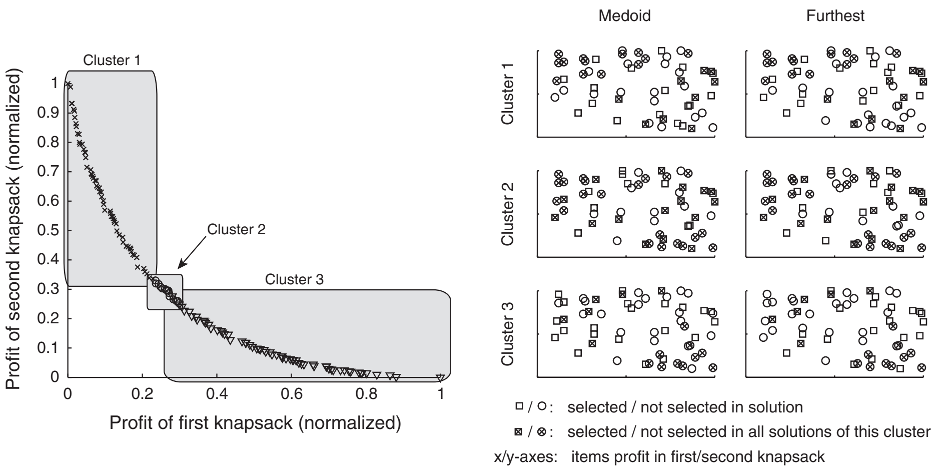

Figure 6. Left: Chosen partitioning with its three clusters (crosses, circles and squares) in objective space. Right: For each cluster (rows), the medoid and the solution furthest from the medoid are plotted. Each plot shows the knapsack items selected/not selected in that specific solutions, plus the items selected/not selected in all solutions of that cluster.

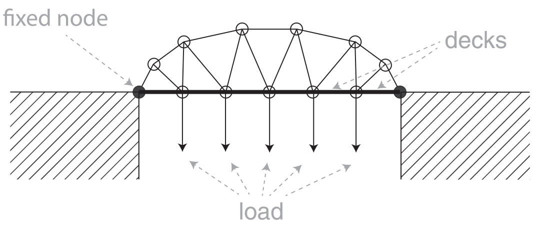

Figure 7. Example bridge. One of the two fixed nodes, the five applied loads and two of the six decks are indicated using dotted arrows.

which the traffic goes. The traffic is modelled as a fixed load that is applied to the five non-fixed nodes between these decks, shown as arrows in the figure. In this paper, we assume that a good bridge will be symmetric. Therefore, we reduce the search space to symmetric bridges only, that is, bridges whose left half is identical to the mirrored right half of the bridge. We do so by starting with randomized symmetric bridges and then ensuring that each change we apply to the bridge is also mirrored to the other side.

The optimization algorithm needs to be able to modify existing bridges in order to create new bridges from old ones. So how do we represent and modify existing bridges? We here use a so-called direct representation, that is, the bridges are directly stored as a set of nodes, and a list of pairs of nodes between which there is a connection. To create new bridges from existing ones, each bridge can be modified through mutation, or two bridges can be recombined. To do mutation, either the nodes or the connections of the bridge can be modified. If a node is modified, three elementary operations can be made: a node can be added, removed or moved. If a node is removed, the node is deleted from the node list, and all connections to or from that node are also deleted. If a node is added, an existing connection (not the decks though) is randomly selected and split by adding a node somewhere in between the end nodes of the connection, removing the old connection and then reconnecting both end nodes to the newly inserted node by inserting two new connections. To move a node, a random node is selected and moved both in horizontal and vertical directions by adding a random number distributed according to a two-dimensional Gaussian distribution. Modifying connections is straightforward. Either a random existing connection (except the decks) is selected and removed or a connection is added between two nodes that have not yet been connected.

To recombine two parent bridges, we use an adaptation of one-point crossover, which is illustrated in Figure 8. Note that because bridges are symmetric, the one-point crossover is actually a two-point crossover with mirrored cut points. First, a cut position is chosen randomly, shown as a vertical line. A second cut is calculated by mirroring the first cut. Both parent bridges are cut at those two positions, and the parts between the cuts are swapped in order to generate two offspring. Hence, the first offspring bridge for example consists of the outer part of the first parent bridge and the inner part of the second parent bridge. Now, there might have been certain connections in both parent bridges that have been destroyed by the cutting. For each connection that was cut, one end

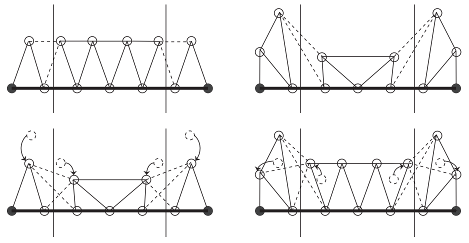
Figure 8. Example for the recombination of two parent bridges (upper row). Cuts are shown as vertical lines. Connections that will be destroyed by the cut are shown as dashed lines in the parents. For both offspring (bottom row), four connections have been cut and need to be reinserted. The corresponding inserted connections are shown as dashed lines in the children. Also, the original nodes (dashed circles) are shown, as well as the offspring nodes to which they are closest (indicated by arrows).

node is retained in the offspring, whereas the other end node is not there anymore. To repair such a connection, all available nodes in the offspring are considered, and the one node that is closest to the removed end node (the one that is not available anymore in the offspring) will be used as the new end node of the connection.

The optimization algorithm also needs to be able to create random bridges to generate the initial population. As randomly generating nodes and connecting them in a random manner is likely to lead to instable bridges, we here propose the following approach. We always start with a (stable) Warren truss, and then we randomly move the nodes of the bridge's top horizontal connections in order to introduce some variation. A warren truss (left) and a random bridge generated from it (right) are shown in the upper row of Figure 8. If a random bridge is unstable, new bridges are generated repeatedly until a stable one is found.

Bridges are evaluated according to two criteria: the weight and the length of the longest connection. We here assume that nodes are weight free, and the total weight is solely determined by the weights of the connections. We chose the bridge weight as the first objective because under a few assumptions, the weight relates linearly to the cost of the bridge through the material cost. These assumptions are that there are no additional cost for nodes and no fixed cost for each connection. The weight of the bridge is calculated as follows. First, it is checked whether the bridge is stable. To do so, we use an approach presented by Rahami *et al*. (2008).[1] If the bridge is not stable, it is discarded. If it is stable, the force on each connection is calculated. Then, the minimum diameter of each connection is calculated. It is chosen such that the connection can withstand the force applied to it, a decision that only depends on the material's yield strength. Now, the weight of the connection can be calculated using this diameter, the length of the connection and the density of the chosen material. The second objective is the length of the longest connections. We chose this objective because in a real-world scenario, long connections might be more difficult to transport than short connections, and they may be difficult to produce.

As mentioned in Section 2, a distance measure in decision space is needed, such that similar-looking bridges have a low distance and dissimilar-looking bridges have a large distance. But how can the distance between two bridges be measured? We here decided to go for a visual measure, on the basis of the shape of each bridge. We define the shape of a bridge as the area enclosed by its outermost connections; see the left plot in Figure 9 for an example. The area difference between the shapes of two bridges then is the distance between the two bridges.

To generate a set of optimized bridges, we used the following specifications. The bridges must support six decks, where each deck has a length of 10 m. Therefore, the bridge has to cover a distance of 60 m. For the load, we assume that the bridge must be able to carry two 40-t trucks; a load of 80 t therefore is applied to each node between the decks. The bridge has to use steel as a material, with a yield strength of 400 MPa, an elasticity (Young's modulus) of 300 GPa and a density of 7.8 g/cm^3. For the variation process in the evolutionary algorithm, we use a recombination probability of 0.7 and a mutation probability of 1.0, and during mutation, we randomly select with equal probability one of the elementary mutations, that is, add a connection, remove a connection, add a node, remove a node or move a node. When adding a node, as explained before, an existing connection is removed and replaced by a new node that is connected to the end nodes of the removed connection. The location of the inserted node is chosen randomly in the rectangle area spanned by the end nodes of the removed connection. When moving a node, a Gaussian is added in each dimension with mean zero and standard deviation three. Whenever infeasible bridges are generated during mutation, we use a repeat strategy that repeatedly tries to do the selected elementary mutation on the parent until either a maximal number of tries, in our case 100, is reached (in which case the parent is returned)

[1]Rahami's Matlab code, which we used in this paper, is available at http://www.mathworks.com/matlabcentral/fileexchange/14313-truss-analysis

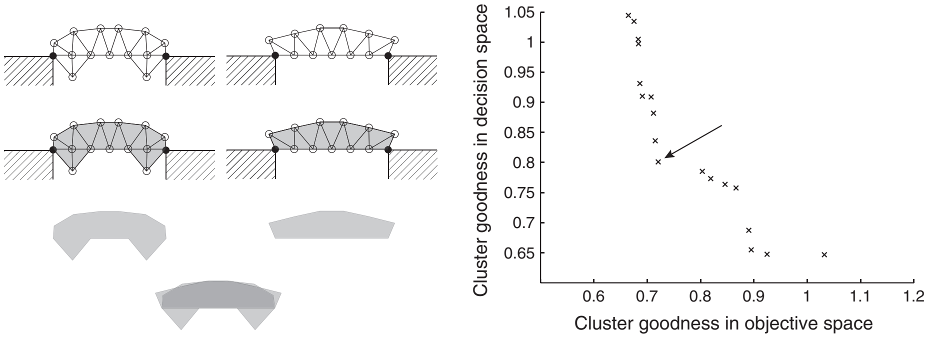

Figure 9. Left: Distance calculation between two bridges (top row). Bridge areas are shown in grey. The distance is visualized in the bottom row, as the lighter grey area. Right: Partitionings found by PAN on the bridge dataset (measures have to be minimized).

or a feasible bridge is found. Furthermore, we use a repair strategy prior to the distance calculation, which iteratively removes all connections on which there is no force and all nodes that are the end nodes of less than two connections.

Using these specifications, we generated an optimized set using the DIOP algorithm (Ulrich *et al.*, 2010) for 100 000 function evaluations and with a population size of 100. We chose DIOP because using a standard MOEA leads to very similar-looking bridges that are not interesting to cluster. DIOP on the other hand optimizes the bridges for structural diversity while having constraints on the bridge's objective values. After deleting duplicates, that is, bridges with decision or objective space[2] distance zero, 98 solutions remain for the partitioning. Note that these bridges are optimized for diversity; that is, there should be no natural clusters of bridges. Therefore, the clustering task actually is very hard, and no trivial partitioning can be expected. The different indices handle the situation in their own way. The S index was not tested as it takes considerably longer to compute than the other three indices. Also, the AS index, which has been proposed to solve the speed problem of the S index, can be used instead. The AS index itself handles the problem of a non-trivial dataset by generating a large number of small clusters. The CS index, on the other hand, tends to find a few very good small clusters and one large cluster that contains the remaining bridges. The DB index is even more extreme and generates the minimum number of clusters, that is, two, where one cluster is very large and the other very small. We will here show only the results of the CS index.

We clustered all the bridges in the given set with the use of direct representation and the CS index, with a population size of 20 and for 80 000 function evaluations. The resulting partitionings are shown in the right plot of Figure 9. In the following text, one of the partitionings, indicated with an arrow in the figure, will be inspected more closely. The chosen partitioning is shown in Figure 10 and consists of a total of eight clusters, of which seven are small clusters with either two or three bridges and one large cluster containing all the remaining bridges. When inspecting the small clusters it can be seen that they indeed contain very similar-looking bridges (one example is given in the upper right corner of the figure). The large cluster, on the other hand, contains many different-looking bridges, although without the most distant-looking bridges. Apparently, these bridges could not be put into smaller clusters without impeding the CS measure. When inspecting the covered objective space area of the smaller clusters, it can be seen that the area they cover is quite different. One extreme is the cluster with a length of the longest connection of 10–12 m and a weight between 800 and 1100 kg. The other extreme is represented by the three clusters that all map to a point in objective space that has a length of the longest connection of approximately 21 and a weight of approximately 580 kg. Overall, a visualization of the whole front as shown in Figure 10 is a much more intuitive way of extracting information from the front than just plotting the objectives or by cluttering the picture with plotting all 98 bridges.

7. CONCLUSIONS

In this paper, we cluster a set of solutions, such that clusters that are compact and well separated in both decision and objective spaces are generated. To

[2]Duplicates in objective space are also deleted as some cluster validity indices cannot handle duplicates in either space.

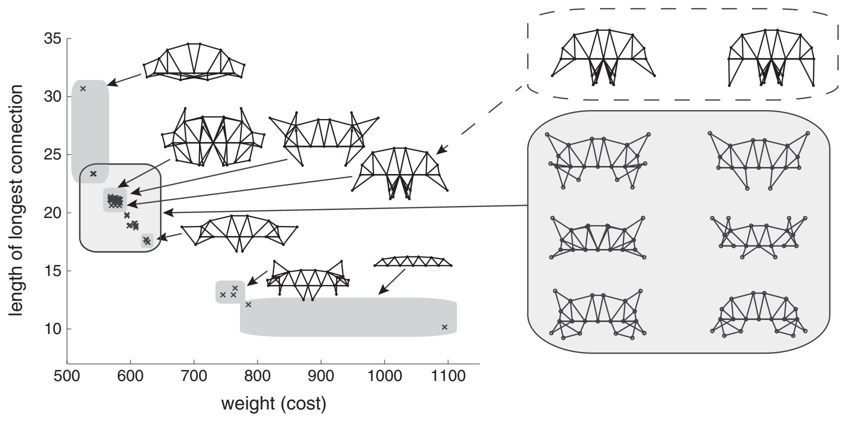

Figure 10. Partitioning achieved by PAN. The partitioning consists of seven small clusters of two or three bridges each and one large cluster of 83 bridges. The objective space values of the bridges are shown on the left, where the small clusters are indicated by a dark grey box. Each small cluster is represented by a random bridge out of that cluster. For one of the clusters, both bridges are shown (dashed box). The large cluster is indicated by a box with a black edge. Six random bridges out of that cluster are shown on the right lower part.

this end, we formally defined this clustering problem as a biobjective optimization problem and designed PAN, a multiobjective evolutionary algorithm (MOEA), to solve the problem. We tested several standard cluster validity indices for their use as optimization goals and several representations found in the literature to represent a partitioning. Applying all representation/validity index combinations to cluster several artificial datasets with known optimal partitionings helped identifying the strengths and weaknesses of the different representations and validity indices, such that a combination that reliably produces good partitionings could be chosen.

The MOEA approach was then compared with the standard clustering approach of repeatedly using the *k*-medoids clustering algorithm for all possible number of clusters. It has been observed that the partitionings found by the MOEA achieve a higher hypervolume in terms of decision and objective spaces goodness than the partitionings found by the *k*-medoids algorithm. When PAN is applied to a knapsack problem, the relation between selected items and achieved profits could be visualized. Also, the method was applied to a real-world truss bridge optimization problem, where a front containing 98 bridges could be visualized in a compact manner by representing each cluster by a representative bridge and by dividing the objective space into regions to which the particular clusters map. In conclusion, it has been found that the proposed method is able to adequately cluster the solutions, such that the clusters contain similar designs and are located in compact regions in objective space.

In the future, a measure to quantify the goodness of tradeoff should be developed. That way, validity indices could not only be compared according to the extreme Pareto-optimal partitionings (which are either best in decision or objective space) but also according to their tradeoffs between the two extreme partitionings. Also, there might be some user preferences; for example, the user has a maximum number of clusters he or she can handle, or he or she values cluster compactness more than cluster separations. PAN could therefore be adapted to incorporate such preferences. Furthermore, PAN might be extended to provide some help in picking one partitioning out of the set of partitionings that is produced.

REFERENCES

Aittokoski T, Ayramo S, Miettinen K. 2009. Clustering aided approach for decision making in computationally expensive multiobjective optimization. *Optimization Methods and Software* **24**:157–174.

Bader J. 2010. Hypervolume-based search for multiobjective optimization: theory and methods. PhD thesis, ETH Zurich, Switzerland.

Bandaru S, Deb K. 2011. Towards automating the discovery of certain innovative design principles through a clustering-based optimization technique. *Engineering Optimization*, Forthcoming Article.

Bandyopadhyay S, Maulik U. 2001. Nonparametric genetic clustering: comparison of validity indices. *IEEE Transactions on Systems, Man, and Cybernetics* **31**:120–125.

Bezdek JC, Pal NR. 1995. Cluster validation with generalized Dunn's indices. In *ANNES*.

Bushel PR, Wolfinger RD, Gibson G. 2007. Simultaneous clustering of gene expression data with clinical chemistry and pathological evaluations reveals phenotypic prototypes. *BMC Systems Biology* **1**:15.

Calinski T, Harabasz J. 1974. A dendrite method for cluster analysis. *Communications in Statistics - Theory and Methods* **3**:1–27.

Calonder M, Bleuler S, Zitzler E. 2006. Module identification from heterogeneous biological data using multiobjective evolutionary algorithms. In *PPSN*.

Chou C-H, Su M-C, Lai E. 2004. A new cluster validity measure and its application to image compression. *Pattern Analysis and Applications* **7**: 205–220.

Conover WJ. 1999. *Practical Nonparametric Statistics* (3rdedn), John Wiley: Hoboken, USA.

Das S, Abraham A, Konar A. 2009. *Metaheuristic Clustering*. Springer: London, UK.

Davies DL, Bouldin DW. 1979. A cluster separation measure. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **1**:224–227.

Deb K, Srinivasan A. 2006. Innovization: innovative design principles through optimization. Technical report, KanGAL, Indian Institute of Technology Kanpur.

Dunn JC. 1974. Well separated clusters and optimal fuzzy-partitions. *Journal of Cybernetics* **4**:95–104.

Eisen MB, Spellman PT, Brown PO, Botstein D. 1998. Cluster analysis and display of genome-wide expression patterns. *Proceedings of the National Academy of Sciences* **95**:14863–14868.

Falkenauer E. 1998. *Genetic Algorithms and Grouping Problems*. Wiley: Hoboken, USA.

Halkidi M, Vazirgiannis M. 2001. Clustering validity assessment: finding the optimal partitioning of a data set. In *ICDM*.

Halkidi M, Vazirgiannis M, Batistakis Y. 2000. Quality scheme assessment in the clustering process. In *PKDD*.

Halkidi M, Batistakis Y, Vazirgiannis M. 2002. Clustering validity checking methods: part II. *SIGMOD Record* **31**:19–27.

Handl J. 2005. Multiobjective clustering around medoids. In *CEC*.

Handl J, Knowles J. 2005a. Improvements to the scalability of multiobjective clustering. In *CEC*.

Handl J, Knowles J. 2005b. Exploiting the trade-off: the benefits of multiple objectives in data clustering. In *EMO*.

Handl J, Knowles J. 2007. An evolutionary approach to multiobjective clustering. *IEEE Transactions on Evolutionary Computation* **11**:56–76.

Houle M, Kriegel H-P, Kröger P, Schubert E, Zimek A. 2010. Can shared-neighbor distances defeat the curse of dimensionality? In *Scientific and Statistical Database Management*. Springer: Berlin, Heidelberg.

Hruschka ER, Campello RJGB, de Castro LN. 2006. Evolving clusters in gene-expression data. *Information Sciences* **176**:1898–1927.

Hruschka ER, Campello RJGB, Freitas AA, de Carvalho ACPLF. 2009. A survey of evolutionary algorithms for clustering. *IEEE Transactions on Systems, Man, and Cybernetics* **39**:133–155.

Jornsten R, Vardi Y, Zhang C-H. 2002. A robust clustering method and visualization tool based on data depth. In *In Statistical data analysis based on the L1norm and related methods*.

Kaufman L, Rousseeuw PJ. 1987. Clustering by means of medoids. Reports of the Faculty of Mathematics and Informatics.

Kaufman L, Rousseeuw PJ. 1990. *Finding Groups in Data: An Introduction to Cluster Analysis*. Wiley: Hoboken, USA.

Kundu D, Suresh K, Ghosh S, Das S, Abraham A, Badr Y. 2009. Automatic clustering using a synergy of genetic algorithm and multi-objective differential evolution. In *HAIS*.

Kutalik Z, Beckmann JS, Bergmann S. 2008. A modular approach for integrative analysis of large-scale gene-expression and drug-response data. *Nature Biotechnology* **26**:531–539.

MacQueen JB. 1967. Some methods for classification and analysis of multivariate observations. In *Proc. of the 5th Berkeley Symposium on Mathematical Statistics and Probability*.

Milligan GW, Cooper MC. 1985. An examination of procedures for determining the number of clusters in a data set. *Psychometrika* **50**:159–179.

Morse JN. 1980. Reducing the size of the nondominated set: pruning by clustering. *Computers and Operations Research* **7**:55–66.

Narayanan M, Vetta A, Schadt EE, Zhu J. 2010. Simultaneous clustering of multiple gene expression and physical interaction datasets. *PLoS Computational Biology* **6**(4).

Obayashi S, Sasaki D. 2003. Visualization and data mining of Pareto solutions using self-organizing map. In *EMO*.

Park YJ, Song MS. 1998. A genetic algorithm for clustering problems. In *Proceedings of the 3rd Annual Conference on Genetic Programming*.

Pollard KS, van der Laan MJ. 2002. Statistical inference for simultaneous clustering of gene expression data. *Mathematical Biosciences* **176**:99–121.

Pryke A, Mostaghim S, Nazemi A. 2006. Heatmap visualization of population based multi objective algorithms. In *EMO*.

Rahami H, Kaveh A, Gholipour Y. 2008. Sizing, geometry and topology optimization of trusses via force method and genetic algorithm. *Engineering Structures* **30**:2360–2369.

Rosenman MA, Gero JS. 1985. Reducing the Pareto optimal set in multicriteria optimization. *Engineering Optimization* **8**:189–206.

Rousseeuw P. 1987. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics* **20**:53–65.

Rudolph G, Naujoks B, Preuss M. 2007. Capabilities of EMOA to detect and preserve equivalent Pareto subsets. In *EMO*.

Sheng W, Liu X, Fairhurst M. 2008. A niching memetic algorithm for simultaneous clustering and feature selection.

IEEE Transactions on Knowledge and Data Engineering **20**:868–879.

Sugimura K, Jeong S, Obayashi S, Kimura T. 2009. Kriging-model-based multi-objective robust optimization and trade-off-rule mining using association rule with aspiration vector. In *CEC*.

Taboada HA, Coit DW. 2007. Data clustering of solutions for multiple objective system reliability optimization problems. *Quality Technology and Quantitative Management* **4**:191–210.

Ulrich T, Brockhoff D, Zitzler E. 2008. Pattern identification in Pareto-set approximations. In *GECCO*.

Ulrich T, Bader J, Thiele L. 2010. Defining and optimizing indicator-based diversity measures in multiobjective search. In *PPSN*.

Xu R, Wunsch DC. 2009. *Clustering*. Wiley: Hoboken, USA.