Representation of the non-dominated set in biobjective discrete optimization

Daniel Vaz, Luís Paquete, Carlos Fonseca, Kathrin Klamroth, Michael Stiglmayr

▶ **To cite this version:**

Daniel Vaz, Luís Paquete, Carlos Fonseca, Kathrin Klamroth, Michael Stiglmayr. Representation of the non-dominated set in biobjective discrete optimization. Computers and Operations Research, 2015, 63, pp.172-186. 10.1016/j.cor.2015.05.003 . hal-04445555

HAL Id: hal-04445555
https://hal.science/hal-04445555

Submitted on 8 Feb 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Representation of the non-dominated set in biobjective discrete optimization

Daniel Vaz[a], Luís Paquete[b,*], Carlos M. Fonseca[b], Kathrin Klamroth[c], Michael Stiglmayr[c]

[a] *Max Planck Institute for Informatics, Saarbrücken, Germany*
[b] *CISUC, Department of Informatics Engineering, University of Coimbra, Portugal*
[c] *Department of Mathematics and Natural Sciences, University of Wuppertal, Germany*

Abstract

This article introduces several algorithms for finding a representative subset of the non-dominated point set of a biobjective discrete optimization problem with respect to uniformity, coverage and the ϵ-indicator. We consider the representation problem itself as multiobjective, trying to find a good compromise between these quality measures. These representation problems are formulated as particular facility location problems with a special location structure, which allows for polynomial-time algorithms in the biobjective case based on the principles of dynamic programming and threshold approaches. In addition, we show that several multiobjective variants of these representation problems are also solvable in polynomial time. Computational results obtained by these approaches on a wide range of randomly generated point sets are presented and discussed.

Keywords: multiobjective discrete optimization, representation, dynamic programming, threshold algorithm

1. Introduction

Typically, multiobjective optimization problems are solved according to the Pareto principle of optimality: A solution is called *efficient* if there is no other feasible solution which is at least as good in all objectives and strictly better in at least one of them. Each of these efficient solutions corresponds to a good compromise among a number of alternatives, and each of them is potentially relevant to a decision maker. Therefore, the goal of multiobjective optimization is to compute the efficient set, from which the decision maker chooses the most preferable solution. Since this set may be too large to present to a decision maker, procedures that produce succinct representations of the efficient set are of particular interest in the context of practical applications.

This article focuses on the computation of a *good* representation of the efficient set, the so called *representation problem*, where the quality of the representation is measured with respect to some property of interest [1][2]. The three underlying assumptions are: i) the efficient set is given; ii) the decision maker is able to choose the preferred solution based solely on its location in the objective space (an efficient solution corresponds to a *non-dominated point* in the objective space); and iii) the cardinality of the representation (k) is provided. Two widely accepted ways of measuring the quality of a representation [3][4][5][6] were introduced by Sayin [2], and are explored in this article: i) *uniformity*, the representation points are as spread as possible; and ii) *coverage*, the representation points are close to the remaining non-dominated points. As these two quality measures may be conflicting, the representation problem that arises from the combination of these two properties, as two objectives, is also of interest.

*Corresponding author
Email addresses: ramosvaz@mpi-inf.mpg.de (Daniel Vaz), paquete@dei.uc.pt (Luís Paquete), cmfonsec@dei.uc.pt (Carlos M. Fonseca), klamroth@math.uni-wuppertal.de (Kathrin Klamroth), stiglmayr@math.uni-wuppertal.de (Michael Stiglmayr)

©2015. Manuscript available under CC-BY-NC-ND 4.0 license
http://creativecommons.org/licenses/by-nc-nd/4.0/
doi: 10.1016/j.cor.2015.05.003 (http://dx.doi.org/10.1016/j.cor.2015.05.003) April 8, 2015

	Dynamic programming	Threshold algorithm
Uniformity	$\mathcal{O}(kn + n \log n)$	$\mathcal{O}(n^2 \log n)$
Coverage	$\mathcal{O}(kn + n \log n)$	$\mathcal{O}(n^2 \log n)$
ϵ-indicator	$\mathcal{O}(kn + n \log n)$	$\mathcal{O}(n^2 \log n)$
Coverage – uniformity	$\mathcal{O}(kn^4 \log n)$	$\mathcal{O}(n^4)$
ϵ-indicator – uniformity	$\mathcal{O}(kn^4 \log n)$	$\mathcal{O}(n^4)$
Coverage – ϵ-indicator	$\mathcal{O}(kn^4 \log n)$	$\mathcal{O}(n^3 \log n)$
Coverage – ϵ-indicator – uniformity	$\mathcal{O}(kn^6 \log n)$	$\mathcal{O}(n^6)$

Table 1: Time-complexities of the dynamic programming and threshold algorithm for the considered representation problems.

A third quality measure, known as the *ϵ-indicator* [7], is also considered in this article. The ϵ-indicator is a widely accepted measure of performance of heuristic approaches to multiobjective optimization. In this article, this indicator is used as a measure of the quality of the representation, and is shown to be closely related to the notion of coverage.

This article considers only representation problems for biobjective combinatorial optimization problems, for which polynomial-time algorithms can be derived for all three quality measures. Since the set of all non-dominated points can be totally ordered in the biobjective case, the representation problem can be recast as a special type of one-dimensional facility-location problems, such as the k-center and k-dispersion problem, for which certain properties can be efficiently explored from an algorithmic point of view.

The three quality measures considered give rise to different representation problems with bottleneck objective functions. Bottleneck objective functions [8] represent a class of objective functions whose goal is to maximize (over all subsets) a minimum quantity over all the elements, or vice-versa. The algorithms proposed in this article fall into two typical types of approaches to this class of problems: dynamic programming and threshold approaches. Whereas dynamic programming consists of solving a sequence of smaller optimization problems, threshold approaches solve a sequence of related feasibility problems. The various algorithms are designed so that they can be easily integrated to solve the representation problems that arise when two or more quality measures are considered together. Table 1 shows the time-complexities that were achieved with the two approaches described in this article for solving several representation problems, where n is the cardinality of the non-dominated set and k is the cardinality of the representation.

The article is organized as follows. Section 2 introduces definitions and notation that will be used throughout the article. Sections 3 to 5 introduce the three representation problems according to uniformity, coverage and the ϵ-indicator, respectively, as well as the two solution methods and experimental results. Then, Section 6 introduces four multiobjective formulations of the representation problem. Finally, Section 7 presents a discussion and conclusions about the main results of this article.

2. Definitions and Notation

In this section, we describe optimality concepts of biobjective optimization problems, as well as several formulations of representation problems that arise from three proposed representation quality measures: uniformity, coverage and ϵ-indicator.

2.1. Optimality concepts

We consider biobjective, discrete optimization problems with two maximizing objectives, that is

$$\operatorname*{vmax}_{x \in X} f(x) = \operatorname*{vmax}_{x \in X} \left(f^1(x), f^2(x) \right) \tag{1}$$

where X denotes the set of feasible solutions and $f^i : X \to \mathbb{R}$, $i = 1, 2$, are two (generally conflicting) objective functions. Let $x, x' \in X$. In the context of Pareto optimality, we introduce the following dominance relations [9]:

- $f(x) \geq f(x')$, i. e. $f(x)$ *weakly dominates* $f(x')$, if and only if $f^i(x) \geq f^i(x'), i = 1, 2$;
- $f(x) > f(x')$, i. e. $f(x)$ *dominates* $f(x')$, if and only if $f(x) \geq f(x')$ and $f(x) \neq f(x')$.

These definitions immediately generalize to higher dimensional problems and to minimization problems. When neither $f(x) > f(x')$ nor $f(x') > f(x)$ holds, we say that the objective vectors $f(x)$ and $f(x')$ are *incomparable*. We also use the same notation among solutions, when the corresponding relation holds in the objective function space. A solution $x \in X$ is called *efficient* (or *Pareto optimal*), if and only if there is no other feasible solution $x' \in X$ such that $f(x') > f(x)$; in this case its corresponding objective vector is called *non-dominated*. The set of all efficient solutions is called the *efficient set* and the set of all non-dominated vectors is called the *non-dominated set* (represented as $\text{vmax}_{x \in X} f(x)$ in Eq. (1)) Note that the vectors in the non-dominated set are pairwise incomparable.

Throughout this article we assume that the feasible set X and its image $Y = f(X) \subset \mathbb{R}^2$ are discrete, and that the non-dominated set $B \subseteq Y$ is finite. We search for a good representation $R \subseteq B$ of the non-dominated set, where different quality measures are used to distinguish between different representations. In the following sections, we assume that the non-dominated set B is known and that a positive integer k with $k \leq n = |B|$ is given. According to some measure of quality, we try to find a *representative* subset $R \subseteq B$ with $|R| = k$. Without loss of generality we assume that all points in B have only positive components.

2.2. Uniformity

Sayin [2] proposed the property of *uniformity* to measure how far apart the k chosen elements of a set R are from each other, or how well they are spread over the non-dominated set. It is computed as the minimum distance between a pair of distinct elements as

$$I_U(R) = \min_{\substack{r_i, r_j \in R \\ r_i \neq r_j}} ||r_i - r_j||, \tag{2}$$

where $||r_i - r_j||$ is a p-norm with $1 \leq p \leq \infty$. The goal of the *uniformity representation problem* is to find a subset R, with a given cardinality k, from a set B that maximizes $I_U(R)$, that is,

$$\max_{\substack{R \subseteq B \\ |R| = k}} I_U(R). \tag{UR}$$

Note that this problem corresponds to a particular case of the k-dispersion problem in facility-location; see, for example, Ravi et al. [10].

2.3. Coverage

A second property proposed by Sayin [2] is *coverage*, which measures the quality of the representative subset by considering the distance of the unchosen elements to their closest elements in the subset. Formally, the coverage of a subset R with respect to a set B is computed as

$$I_C(R, B) = \max_{b \in B} \min_{r \in R} ||r - b||.$$

The *coverage representation problem* consists of finding the subset of cardinality k that has the smallest coverage value (coverage radius), that is,

$$\min_{\substack{R \subseteq B \\ |R| = k}} I_C(R, B). \tag{CR}$$

This problem is known in the literature of facility-location as the k-center problem; see Kariv and Hakimi [11] for an early reference as well as Hassin and Tamir [12] and Schöbel [13] for a problem closely related to the coverage representation problem considered here.

2.4. ϵ-indicator

The ϵ-indicator is a well known measure of performance in heuristic solution approaches and approximation algorithms for multiobjective optimization problems [14]. In the context of the representation problem considered here, it corresponds to the smallest factor that can be multiplied to each element in the subset R such that every point in the set B becomes weakly dominated. Therefore, it is related to the notion of ϵ-dominance. The ϵ-indicator is calculated as follows [7]:

$$I_\epsilon(R, B) = \max_{b \in B} \min_{r \in R} \epsilon(r, b),$$

where

$$\epsilon(r, b) = \max_{i \in \{1,2\}} (b^i / r^i),$$

i. e. $\epsilon(r, b)$ is the smallest factor such that $b \leq \epsilon(r, b) \cdot r$ (recall that we consider a biobjective maximization problem). Note that if $R = B$, we have that $I_\epsilon(R, B) = 1$. Also note that the factor $\epsilon(r, b)$ defined above corresponds to the approximation ratio $(1 + \epsilon)$ typically used in the context of approximation schemes; see, for example, Papadimitriou and Yannakakis [14]. The *ϵ-indicator representation problem* consists of finding the subset R of cardinality k with the smallest ϵ-indicator, that is,

$$\min_{\substack{R \subseteq B \\ |R|=k}} I_\epsilon(R, B). \tag{ER}$$

Note that this problem is closely related to the coverage representation problem in the sense that a good coverage usually implies a good ϵ-indicator, and vice versa. The algorithms developed for these two representation problems reflect this similarity. More precisely, we will show in the following sections that the coverage and the ϵ-indicator problem can be solved by very similar algorithms, however, using different, measure dependent adjacency data.

We remark that, even though they share similar structure and properties, the choice of indicator affects the representation that is obtained. In particular, while ϵ-indicator is non-uniform scale invariant, coverage is translation invariant. Furthermore, ϵ-indicator relates to the concept of increase by a certain percentage, whereas coverage introduces a local perspective, for which representations correspond to clusters of points. These properties can be used to easily generate instances where the best representations for coverage and ϵ-indicator will be different, for instance, by considering much larger numbers along one of the coordinates.

2.5. Multiobjective representation problems

In addition to the three representation problems defined in Sections 2.2–2.4 above, we consider combinations of the three quality measures, uniformity, coverage and ϵ-indicator, and analyse their respective trade-offs. In this context, we consider multiobjective representation problems that arise from the combination of these three quality measures.

The goal of the biobjective *coverage–uniformity* representation problem is to find subsets $R \subseteq B$ of cardinality k that minimize the coverage and simultaneously maximize the value of uniformity. For consistency, we invert the sign of the uniformity value. Therefore, we have the following problem formulation:

$$\operatorname*{vmin}_{\substack{R \subseteq B \\ |R|=k}} \left(I_C(R, B), -I_U(R)\right). \tag{CUR}$$

The representation problems considered are multiobjective minimization problems. Therefore, the inequalities for dominance change to reflect minimization, that is:

- $f(x) \leq f(x')$, i. e. $f(x)$ *weakly dominates* $f(x')$, if and only if $f^i(x) \leq f^i(x'), i = 1, 2$;
- $f(x) < f(x')$, i. e. $f(x)$ *dominates* $f(x')$, if and only if $f(x) \leq f(x')$ and $f(x) \neq f(x')$.

The remaining notation is defined analogously to the maximization version. In particular, vmin refers to the non-dominated points of this multiobjective minimization problem in terms of Pareto optimality.

We now formulate the *ϵ-indicator–uniformity* and the *coverage–ϵ-indicator* representation problems analogously:

$$\operatorname*{vmin}_{\substack{R\subseteq B\\|R|=k}} \left(I_\epsilon(R,B), -I_U(R)\right), \tag{EUR}$$

$$\operatorname*{vmin}_{\substack{R\subseteq B\\|R|=k}} \left(I_C(R,B), I_\epsilon(R,B)\right). \tag{CER}$$

The three-objective version of the representation problem aims at minimizing both the coverage and the ϵ-indicator, while maximizing the uniformity. Formally, this problem is defined as follows:

$$\operatorname*{vmin}_{\substack{R\subseteq B\\|R|=k}} \left(I_C(R,B), I_\epsilon(R,B), -I_U(R)\right). \tag{CEUR}$$

2.6. General properties

The two quality measures, uniformity and coverage, are based on distances between pairs of points in the non-dominated set B. Since we consider biobjective problems (i. e., $B \subseteq \mathbb{R}^2$), the points in B can be ordered in such a way that their first component is strictly increasing, while their second component is strictly decreasing. This is a very strong property (which is in general not satisfied in higher-dimensional problems) that is extremely useful when comparing inter-point distances in the context of these two quality measures.

Proposition 2.1. *Let the vectors $a, b, c \in B \subset \mathbb{R}^2$ be pairwise incomparable, and suppose that they are sorted with respect to their first component, i. e., $a^1 < b^1 < c^1$. Then the following inequality holds for all p-norms ($1 \leq p \leq \infty$):*

$$||a - c|| > ||b - c||.$$

Proof. Since a, b, c are non-dominated and thus pairwise incomparable, we have that $a^2 > b^2 > c^2$. Therefore, $c^1 - a^1 > c^1 - b^1$, and since both sides of the inequality are positive, $|c^1 - a^1|^p > |c^1 - b^1|^p$ for all $1 \leq p < \infty$. Similarly, $a^2 - c^2 > b^2 - c^2$ and hence $|a^2 - c^2|^p > |b^2 - c^2|^p$ for all $1 \leq p < \infty$. For $1 \leq p < \infty$, we can use the fact that the p-th root is a non-decreasing function, and obtain

$$||a - c||_p = \sqrt[p]{|a^1 - c^1|^p + |a^2 - c^2|^p} > \sqrt[p]{|b^1 - c^1|^p + |b^2 - c^2|^p} = ||b - c||_p.$$

For the ∞-norm, $|c^1 - a^1| > |c^1 - b^1|$ and $|a^2 - c^2| > |b^2 - c^2|$. We conclude that

$$||a - c||_\infty = \max\{|a^1 - c^1|, |a^2 - c^2|\} > \max\{|b^1 - c^1|, |b^2 - c^2|\} = ||b - c||_\infty.$$

□

Corollary 2.2. *Let the vectors $b_1, b_2, \ldots, b_n \in B \subset \mathbb{R}^2$ be pairwise incomparable, and suppose that they are sorted with respect to their first component, i. e., $b_1^1 < b_2^1 < \ldots < b_n^1$. Then the distances of the non-dominated points $b_2, \ldots, b_n$ to the lexicographic minimum b_1 are strictly increasing, i. e., $||b_2 - b_1|| < ||b_3 - b_1|| < \ldots < ||b_n - b_1||$, for all p-norms ($1 \leq p \leq \infty$).*

The inter-point distances in the sorted set of non-dominated points in $B \subset \mathbb{R}^2$ thus satisfy the same pairwise inequalities as if the points were aligned on a one-dimensional structure (a line) in $\mathbb{R}^2$. We will utilize this fact for the efficient computation of optimal representations in the following sections. The algorithmic techniques have some similarities to algorithms for location problems in one dimension, see, for example, [10, 12, 13]. Even though the ϵ-indicator is not directly defined based on inter-point distances, a similar property also holds in this case.

Proposition 2.3. *Let the vectors $a, b, c \in B \subset \mathbb{R}^2$ be pairwise incomparable, and suppose that they are sorted with respect to their first component, i. e., $a^1 < b^1 < c^1$. Then the ϵ-indicator satisfies*

$$\epsilon(a,b) < \epsilon(a,c) \quad \textit{and} \quad \epsilon(c,a) > \epsilon(c,b).$$

PROOF. As in the proof of Proposition 2.1, we have that $a^2 > b^2 > c^2$. Hence, $\frac{b^1}{a^1} > 1$ and $\frac{b^2}{a^2} < 1$, and thus $\epsilon(a,b) = \max\{\frac{b^1}{a^1}, \frac{b^2}{a^2}\} = \frac{b^1}{a^1}$. Similarly, $\epsilon(a,c) = \frac{c^1}{a^1}$. Since $b^1 < c^1$ we obtain that

$$\epsilon(a,b) = \frac{b^1}{a^1} < \frac{c^1}{a^1} = \epsilon(a,c).$$

The second inequality is proven analogously. □

Corollary 2.4. *Let the vectors $b_1, b_2, \ldots, b_n \in B \subset \mathbb{R}^2$ be pairwise incomparable, and suppose that they are sorted with respect to their first component, i. e., $b_1^1 < b_2^1 < \ldots < b_n^1$. Then $\epsilon(b_1,b_2) < \epsilon(b_1,b_3) < \cdots < \epsilon(b_1,b_n)$, and similarly, $\epsilon(b_n,b_1) > \epsilon(b_n,b_2) > \cdots > \epsilon(b_n,b_{n-1})$.*

3. The Uniformity Representation Problem

In this section we describe two algorithms to solve the uniformity representation problem: An adaptation of the dynamic programming algorithm proposed by Wang and Kuo [15] and a threshold algorithm. Numerical results confirm the theoretical advantage of the dynamic programming approach.

3.1. Dynamic programming

The dynamic programming algorithm for problem (UR) is based on the approach proposed by Wang and Kuo [15], which we briefly describe as follows. Assume, without loss of generality, that $B = \{b_1, \ldots, b_n\}$ and that $b_j^1 < b_\ell^1$ implies that $j < \ell$, i. e., the vectors in B are sorted with respect to their first component. Let $S_{i,j}$ ($i \in \{1, \ldots, n\}$, $j \in \{1, \ldots, n-i+1\}$) denote the optimal subset with cardinality i that contains b_j and only elements taken from $\{b_j, \ldots, b_n\}$. Then, for $\ell > j$, the following holds:

$$I_U(\{b_j\} \cup S_{i-1,\ell}) = \min\left\{||b_j - b_\ell||, I_U(S_{i-1,\ell})\right\}.$$

Let T be a $k \times n$ matrix, where each entry $T(i,j)$ stores the optimal uniformity value for the subsets of cardinality i that contain b_j and only elements taken from the set $\{b_j, \ldots, b_n\}$, i. e., $T(i,j) = I_U(S_{i,j})$. The base case is $T(1,j) = +\infty$, $1 \le j \le n$. For $i > 1$, the following recursion is applied:

$$T(i,j) = \max_{j < \ell \le n-i+2} \min\left\{||b_j - b_\ell||, T(i-1,\ell)\right\}. \tag{3}$$

After computing all values in the matrix T, the optimal uniformity value can be obtained as

$$\max_{1 \le j \le n-k+1} T(k,j).$$

As reported by Wang and Kuo [15], it is possible to further improve this algorithm by applying a modification which removes the need to check the optimal value for every subproblem, obtaining a time complexity of $\mathcal{O}(nk + n\log n)$. We present the main results of Wang and Kuo [15] in the following.

Proposition 3.1. *$T(i,j) \ge T(i,j+1)$, for $1 \le i \le n$ and $1 \le j \le n-i$.*

PROOF. Let $S' = (S_{i,j+1} \setminus \{b_{j+1}\}) \cup \{b_j\}$. Given that $T(i,j)$ is the optimal uniformity value for the subsets of cardinality i taken from $\{b_j, \ldots, b_n\}$, then it holds that $T(i,j) \ge I_U(S')$. Therefore, we have the following result:

$$T(i,j) \ge I_U(S') = \min_{\substack{b_p, b_q \in S' \\ b_p \ne b_q}} ||b_p - b_q|| \ \ge \min_{\substack{b_p, b_q \in S_{i,j+1} \\ b_p \ne b_q}} ||b_p - b_q|| = I_U(S_{i,j+1}) = T(i,j+1).$$

□

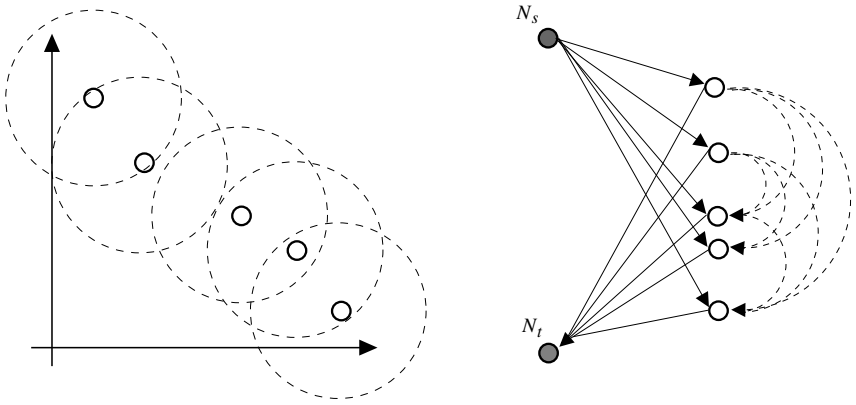

Figure 1: Threshold approach for uniformity problem: Set B with circles of radius u (left) and respective graph G_U (right)

When evaluating the maximization over $\ell \in \{j+1, \ldots, n-i+2\}$ for the determination of $T(i,j)$ in recursion (3), $T(i-1,\ell)$ is non-increasing for increasing ℓ according to Proposition 3.1, and $||b_j - b_\ell||$ is increasing for increasing ℓ, by Corollary 2.2. Thus, the minimum of the two values equals $||b_j - b_\ell||$ up to an index $\bar{\ell} \in \{j, \ldots, n-i+2\}$ (and it is thus increasing up to $\bar{\ell}$), and it equals $T(i-1,\ell)$ for all indices larger than $\bar{\ell}$ (and it is thus non-increasing for $\ell > \bar{\ell}$). Therefore, instead of iterating over all the possible values of ℓ, the algorithm stops when the next value of ℓ does not increase the uniformity value. Therefore, on average, the algorithm checks $\mathcal{O}(1)$ values of ℓ for each pair (i,j). This leads to the time complexity of $\mathcal{O}(k\,n)$ described by Wang and Kuo [15].

3.2. Threshold approach

Threshold algorithms are typical approaches to solve optimization problems with bottleneck objective functions [8]. They consist of solving a sequence of feasibility problems until the optimal solution for the optimization problem is found. Our approach to the uniformity representation problem is based on the threshold algorithm described in Ponte et al. [16] for a related problem with the ϵ-indicator.

The threshold algorithm explores the fact that the optimal value is one of the distances between pairs of elements in B; see equation (2). Let $u_1, u_2, \ldots, u_m$, with $m \leq \frac{n^2}{2} - \frac{n}{2}$, be the list of possible uniformity values, sorted in increasing order. The algorithm searches for the optimal uniformity value by performing a binary search and checking, for each considered value of $u \in \{u_1, \ldots, u_m\}$, if there exists a feasible solution, i. e., a subset with k elements whose uniformity value is at least u. The feasibility problem that we consider consists of finding the subset with maximum cardinality that has a uniformity value of at least u. If the maximum cardinality is larger than k, arbitrary elements can be removed from the subset to form a subset of cardinality k. Note that removing an element from the subset never decreases the uniformity value.

In order to find a subset of maximum cardinality for a given uniformity value u, we transform the feasibility problem into a network optimization problem as follows. We define a directed graph $G_U = (N, E_U)$ with node set N and arc set E_U. The graph contains the following $n+2$ nodes: N_s as a source node, N_t as a target node, and N_i as a node for each element $b_i \in B$, $i = 1, \ldots, n$. The arcs are defined as follows: $(N_s, N_i) \in E_U$ and $(N_i, N_t) \in E_U$, for all $i \in \{1, \ldots, n\}$, and $(N_i, N_j) \in E_U$ if and only if $||b_i - b_j|| \geq u$, $i, j \in \{1, \ldots, n\}$ with $i < j$. Figure 1 illustrates this transformation. The left plot shows the elements of the set B in the objective space as white circles. Each dashed ball is centered at a given point and its radius is u. The right plot shows the resulting graph G_U where the grey circles on the left correspond to the source and target node, respectively, and the white circles correspond to the nodes N_1 to N_5. Note that the dashed arcs link two of the latter nodes if their distance is greater or equal than the uniformity value u.

It is easy to see that there is a bijection between the subsets $R \subseteq B$ such that $I_U(R) \geq u$ and the paths source-to-target in the graph G_U: First observe that each element $b_i \in B$ has a corresponding node $N_i \in N$, and hence there is a direct relation between a subset $R \subseteq B$ and a path from N_s to N_t in G_U (excluding

nodes N_s and N_t). It remains to show that every subset R obtained from a path from N_s to N_t in G_U satisfies $I_U(R) \geq u$. To this end, recall that Proposition 2.1 implies that for any $i < j < \ell$, if there is an arc $(N_i, N_j) \in E_U$, then we have that $||b_i - b_\ell|| \geq ||b_i - b_j|| \geq u$. Thus, if the arc (N_i, N_j) is in the path, and node N_ℓ is a later node in the same path, then b_i and b_ℓ (with $\ell > j$) have a larger distance than the uniformity value u. Therefore, the uniformity value of the related set R satisfies $I_U(R) \geq u$.

By using the graph G_U, we are able to check if there is a feasible subset with a uniformity value of u by finding a corresponding path with at least $k+1$ arcs. Since any path with $k+1$ arcs will suffice, the algorithm finds a longest path in the graph. The longest path is easily found by starting with the first node N_1 and selecting at every node N_i the smallest possible step, i. e., the edge $(N_i, N_{j'})$ with $j' = \min\{j : (N_i, N_j) \in E_U\}$, which has time complexity of $\mathcal{O}(n)$. We now prove that this procedure produces a longest path.

Proposition 3.2. *The greedy procedure described above produces a path of maximal length in the graph G_U.*

Proof. Let P be a path of maximal length.
Claim 1: There is a path $\hat{P}$ of maximal length that contains N_1. If this is not the case, its second node can be replaced by N_1 without changing the length of the path.
Claim 2: Any edge (N_j, N_k) can be replaced by the edge to the closest node connected to N_j. Suppose edge (N_j, N_k) is the first edge in $\hat{P}$ that is not a shortest edge, and let (N_k, N_l) be the next edge in $\hat{P}$. Let furthermore $\tilde{k} = \min_{i \in \{1,\ldots,n\}}\{i : (N_j, N_i) \in E_U\}$ be the closest node to N_j connected by an edge. Then replacing the edges (N_j, N_k) and (N_k, N_l) in $\hat{P}$ by the edges $(N_i, N_{\tilde{k}})$ and $(N_{\tilde{k}}, N_l)$ leads to a path of the same length. Applying this procedure iteratively yields a path as generated by the greedy algorithm. □

Note that similarly, we could apply a standard dynamic programming technique to obtain the longest path for each node, by going through the nodes in topological order. Although this takes $\mathcal{O}\left(n^2\right)$, it has no influence on the time complexity of the complete algorithm, and is in fact more useful for the multiobjective problems described in Section 2.5.

We split the threshold algorithm into two parts: the preprocessing, which consists in sorting the distance values, and whose time complexity we denote by $T_P(n)$, and the search itself, consisting in executing the procedure to solve the longest path problem with time complexity $T_C(n)$, repeated over $T_S(n)$ values of uniformity, resulting in a time complexity of $\mathcal{O}(T_P(n) + T_S(n) \cdot T_C(n))$. By following the structure and improvements proposed by Vaz et al. [17] we have that $T_P(n) = \mathcal{O}\left(n^2 \log n\right)$ and $T_S(n) = \mathcal{O}(\log n)$. The longest path algorithm visits each node once, leading to a time complexity of $T_C(n) = \mathcal{O}(n)$ and an overall time complexity of $\mathcal{O}\left(n^2 \log n\right)$.

3.3. Experimental analysis

We ran two tests with the two approaches in order to study the influence of n and k on the performance of the algorithms. The implementations were tested on randomly generated instances, repeating each instance five times in order to eliminate slight fluctuations in running time. For each pair (n, k) of tested values, we ran 5 different random instances. The points were sampled uniformly at random from a circle centered on the origin intersected with the positive quadrant. We remark that, since the algorithms are deterministic, they always return the same result.

The tests were run in eight similar nodes of a cluster running the Sun Grid Engine for job management. The nodes where the tests were run are equipped with 16 GB of RAM and "Intel(R) Core(TM) i7-3770K" CPUs running at 1600MHz, with four jobs per node. The source code for the implementations was written in C++ and compiled with GCC 4.4.3. C++ was chosen over C for its convenience, since its compiler is less strict. However, the Standard Template Library (STL) and the object-oriented features were not used. The only exception is the use of "vector" to store the solutions in the multiobjective representation problems introduced in Section 2.5.

Figure 2, left plot, shows the results for the first test ($k = 20$) in a log-log scale. The difference of the running time of the threshold algorithm (TA) to the dynamic programming (DP) version increases to a factor of 10^3. We recall that the running time of the dynamic programming algorithm grows linearly with

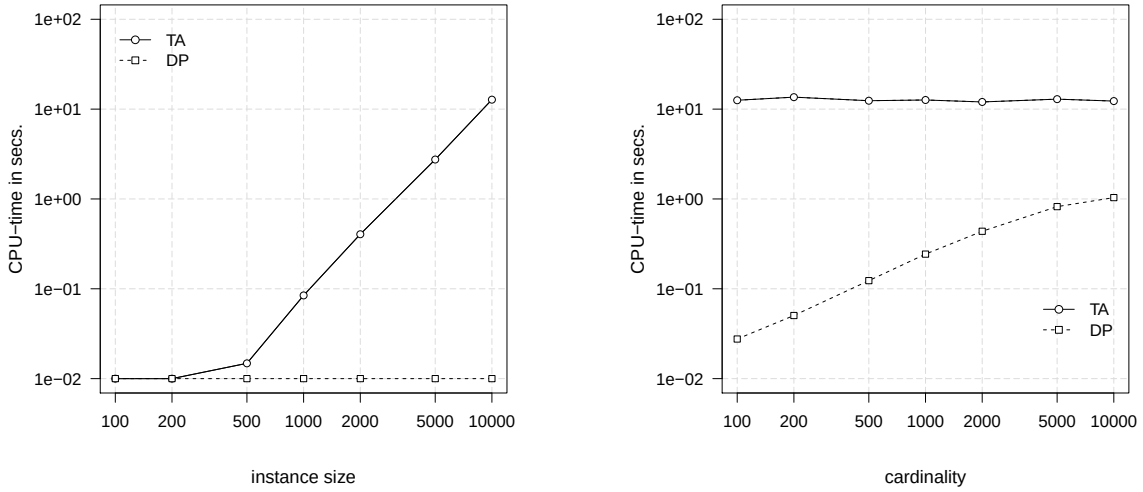

Figure 2: Running time of the dynamic programming and threshold approach for the uniformity problem w. r. t. instance size n on the left (with $k = 20$ fixed) and cardinality k on the right ($n = 10000$)

n, but quadratically for the threshold algorithm. Therefore, it is not surprising that the former clearly outperforms the latter.

In the second test, we studied the influence of k. Since the complexity of the threshold algorithm is not dependent on k, it is expected that the algorithm has constant performance. However, the dynamic programming approach depends on k, and consequently may take more time to solve instances with a larger value of k. The results are presented in Figure 2, right plot. They indicate that, even though the threshold algorithm has a similar running time for different values of k, and that the performance of the dynamic programming approach decreases as k increases, it is still better by an order of magnitude. This can be explained by the fact that k is the size of the representation, i. e., $k \leq n$. Thus, the complexity of the dynamic programming approach, $\mathcal{O}(k\,n)$, is in this worst case $\mathcal{O}\left(n^2\right)$, which is still better than the complexity of the threshold algorithm. Therefore, even if $k = n$, the complexity of the dynamic programming algorithm is better, and consequently it is not surprising that it outperforms the threshold algorithm in our computational tests.

4. The Coverage Representation Problem

Analogous to the case of the uniformity problem discussed in the previous section, we describe a dynamic programming algorithm and a threshold algorithm for the coverage representation problem and compare the two methods in an experimental analysis.

4.1. Dynamic programming

The dynamic programming algorithm for Problem (CR) follows the same working principle as the dynamic programming algorithm for Problem (UR). Moreover, it is closely related to the dynamic programming approaches for the one dimensional k-center problems described in Hassin and Tamir [12] and Schöbel [13]. Let again $B = \{b_1, \ldots, b_n\}$ be sorted with respect to the first component. As introduced in the previous section, we denote as $S_{i,j}$ ($i \in \{1, \ldots, n\}$, $j \in \{1, \ldots, n - i + 1\}$) the optimal representation of cardinality i of the set $\{b_j, \ldots, b_n\}$ that contains b_j and only elements taken from $\{b_j, \ldots, b_n\}$. Then, for $\ell > j$, as a consequence of Corollary 2.2, the following holds:

$$I_C(\{b_j\} \cup S_{i-1,\ell}, \{b_j, \ldots, b_n\}) = \max\left\{\delta_{j,\ell}, I_C(S_{i-1,\ell}, \{b_\ell, \ldots, b_n\})\right\}$$

where

$$\delta_{j,\ell} = \max_{j \le m \le \ell} \min\left\{||b_j - b_m||, ||b_m - b_\ell||\right\}. \tag{4}$$

As in Section 3.1, let T be a $k \times n$ matrix. In each entry $T(i,j)$ the optimal coverage value is stored for the representations of cardinality i of the set $\{b_j, \ldots, b_n\}$ that contain b_j and elements taken from the set $\{b_j, \ldots, b_n\}$ i. e., $T(i,j) = I_C(S_{i,j}, \{b_j, \ldots, b_n\})$. The base case is $T(1,j) = ||b_j - b_n||$ and the recursion is given by

$$T(i,j) = \min_{j < \ell \le n-i+2} \max\left\{\delta_{j,\ell}, T(i-1,\ell)\right\}. \tag{5}$$

In order to extract the optimal coverage value from the last row of the matrix T, the entries $T(k,j)$ must be corrected since, for a given $j > 1$, the elements in $\{b_1, \ldots, b_{j-1}\}$ were not considered in the calculation of $T(k,j)$. From Proposition 2.1, we only need to consider the element b_1. This correction is obtained as follows:

$$\bar{T}(k,j) = \max\left\{||b_1 - b_j||, T(k,j)\right\}.$$

After this correction step, the optimal coverage value is obtained as

$$\min_{1 \le j \le n-k+1} \bar{T}(k,j).$$

For each entry of the matrix T all the possible values of ℓ, which are at most $\mathcal{O}(n)$, must be checked. Moreover, for each value of ℓ we need to check whether b_j or b_ℓ is closer to b_m, for $j < m < \ell$. Since there may be $\mathcal{O}(n)$ values of m, the time complexity for this algorithm is $\mathcal{O}\left(k\, n^3\right)$.

We now show that some improvements can decrease the time complexity of this dynamic programming algorithm.

4.1.1. m-improvement

Recall from Proposition 2.1 that for $j < m < \ell$, $||b_j - b_m|| < ||b_j - b_{m+1}||$ and $||b_m - b_\ell|| > ||b_{m+1} - b_\ell||$ hold. Therefore, as m increases, $||b_j - b_m||$ increases and $||b_m - b_\ell||$ decreases. Since the value of $\delta_{j,\ell}$ is the maximum over the indices m, we can easily compute $\delta_{j,\ell}$ if we know the correct index of m, which we denote by m'_ℓ. By Proposition 2.1, $||b_m - b_\ell|| < ||b_m - b_{\ell+1}||$, which means that if ℓ increases, $||b_m - b_\ell||$ also increases, and hence $m'_\ell \le m'_{\ell+1}$.

Therefore, it is more efficient to simply calculate the values $\delta_{j,\ell}$ by fixing a value of j and iterating through all values of ℓ. For the first value, $\ell = j+1$, we must have $m'_\ell = j$ or $m'_\ell = \ell$. For all other values for ℓ, m'_ℓ must be at least as large as $m'_{\ell-1}$. Therefore, it is sufficient to start with $m'_\ell = m'_{\ell-1}$ and increase m'_ℓ until the maximum value for $\min\left\{||b_j - b_m||, ||b_m - b_\ell||\right\}$ is attained. This allows us to pre-compute all the values for a given j while increasing the index of m at most $\mathcal{O}(n)$ times. Therefore, this procedure has time complexity $\mathcal{O}\left(n^2\right)$ for each row of T, and the overall time complexity becomes $\mathcal{O}\left(k\, n^2\right)$.

4.1.2. ℓ-improvement

Wang et al. [15] suggest a speed-up technique that can be adapted for this algorithm. This improvement removes the need to iterate over values of ℓ as it is possible to test a small number of these values, given the value used previously. We first show that $T(i,j)$ and $\delta_{j,\ell}$ are non-increasing when j increases, and $\delta_{j,\ell}$ is non-decreasing as ℓ increases.

Proposition 4.1. *If $j < \ell$, then $\delta_{j,\ell} \ge \delta_{j+1,\ell}$ and $\delta_{j,\ell} \le \delta_{j,\ell+1}$.*

Proof. From Proposition 2.1 we have that $||b_j - b_m|| > ||b_{j+1} - b_m||$, for $j < j+1 \le m$. Therefore, it holds that

$$\min\left\{||b_{j+1} - b_m||, ||b_m - b_\ell||\right\} \le \min\left\{||b_j - b_m||, ||b_m - b_\ell||\right\}$$

and

$$\delta_{j+1,\ell} = \max_{j+1 \le m \le \ell} \min\left\{||b_{j+1} - b_m||, ||b_m - b_\ell||\right\} \le \max_{j+1 \le m \le \ell} \min\left\{||b_j - b_m||, ||b_m - b_\ell||\right\}.$$

Extending the maximization to the interval $m \in \{j, \ldots, \ell\}$ will not decrease the maximum value, and hence

$$\delta_{j+1,\ell} \leq \max_{j+1 \leq m \leq \ell} \min\{||b_j - b_m||, ||b_m - b_\ell||\} \leq \max_{j \leq m \leq \ell} \min\{||b_j - b_m||, ||b_m - b_\ell||\} = \delta_{j,\ell}.$$

The proof for the second part is analogous. □

Proposition 4.2. *$T(i,j) \geq T(i,j+1)$, for $1 \leq i \leq n$ and $1 \leq j \leq n-i$.*

Proof. Recall that $I_C(S_{i,j}, \{b_j, \ldots, b_n\}) = T(i,j)$. We denote by $B_{p,q}$ the set $\{b_p, \ldots, b_q\}$, $p \leq q$, and by S' the set obtained by replacing b_j by b_{j+1} in the set $S_{i,j}$, that is, $S' = (S_{i,j} \cap B_{j+1,n}) \cup \{b_{j+1}\}$. (Note that if $S_{i,j}$ contains both b_j and b_{j+1}, then $|S'| = i-1$. This has, however, no impact on the subsequent derivations.) Given that $T(i,j+1)$ is the optimal coverage value (with cardinality i) for the subset $B_{j+1,n}$, we know that $T(i,j+1) \leq I_C(S', \{b_{j+1}, \ldots, b_n\})$ holds. From Proposition 2.1 we have that $||b_j - b_m|| \geq ||b_{j+1} - b_m||$, for $j+1 \leq m$. Therefore, using the fact that $S_{i,j}$ and S' only differ by one element (b_{j+1} instead of b_j), and that b_{j+1} has a lower distance to b_m, we have the following result:

$$\begin{aligned} T(i,j+1) \leq I_C(S', B_{j+1,n}) &= \max_{b \in B_{j+1,n}} \min_{s \in S'} ||s - b|| \\ &\leq \max_{b \in B_{j+1,n}} \min_{s \in S_{i,j}} ||s - b|| \\ &\leq \max_{b \in B_{j,n}} \min_{s \in S_{i,j}} ||s - b|| \\ &= I_C(S_{i,j}, B_{j,n}) = T(i,j). \end{aligned}$$

□

In the recursion (5), the value of $T(i,j)$ is calculated as the minimum over $j < \ell \leq n-i+2$ of the maximum of one non-decreasing sequence ($\delta_{j,\ell}$) and one non-increasing sequence ($T(i-1,\ell)$), as shown in Propositions 4.1 and 4.2, respectively. Therefore, the maximum of the two sequences is a bitonic sequence that decreases until the non-increasing sequence becomes smaller than the non-decreasing sequence, and then increases. Moreover, if we now look at the calculation of $T(i,j-1)$, the non-increasing sequence ($T(i,\ell)$) remains equal, but by Proposition 4.1, $\delta_{j-1,\ell} \geq \delta_{j,\ell}$ for all $\ell > j$. Thus, the sequence ($\delta_{j-1,\ell}$) may attain larger values than ($T(i,\ell)$) for a smaller index ℓ than the sequence ($\delta_{j,\ell}$).

Therefore, instead of iterating over all the possible values of ℓ, the algorithm stops when the next value of ℓ increases the coverage value. Furthermore, if we also use the previous improvement, the time complexity of calculating all the values $T(i,j)$ is now $\mathcal{O}(k\,n)$, since on average, the algorithm checks $\mathcal{O}(1)$ values of ℓ. We recall that, by using the previous improvement, we need to add a preprocessing step with time complexity $\mathcal{O}(n^2)$. Therefore, this version of the algorithm has $\mathcal{O}(n^2)$ time complexity.

4.1.3. ℓ-improvement with no preprocessing

One further improvement, based on the previous ones, is to keep the idea of using previous values for ℓ and m, without needing the pre-processing step. The way we do this is to, each time we increase the value of ℓ (as described in Section 4.1.2), also increase the value of m, as described in Section 4.1.1. However, neither of the sections describes how should we update the value of m when j is increased.

This problem is solved by once again considering a similar approach. By Proposition 2.1, $||b_j - b_m|| > ||b_{j+1} - b_m||$, which means that if j increases, $||b_j - b_m||$ also increases, and hence the corresponding value of m also increases. Therefore, by increasing m each time j or ℓ increases, until we reach the maximum value of $\delta_{j,\ell}$, we are able to efficiently update m, increasing it at most $\mathcal{O}(n)$ times during the whole algorithm.

Using this improvement, we no longer need to pre-compute the values of m, and the time complexity of calculating all the values $T(i,j)$ is still $\mathcal{O}(k\,n)$. Therefore, this improved version of the algorithm has time complexity $\mathcal{O}(k\,n)$.

4.2. Threshold approach

Threshold approaches ask for an efficient solution of the corresponding feasibility problem. In the following, we present two approaches to determine a representation with a given coverage value, the first based on a set-cover formulation and the second based on a shortest path formulation.

The distances between pairs of points are the possible values for the optimal coverage value. In a preprocessing step, these distances are computed and sorted in increasing order $c_1 < c_2 < \cdots < c_m$ with $m \leq \frac{n^2}{2} - \frac{n}{2}$. Then, the algorithm performs a binary search on this list and checks whether a feasible solution exists for a given threshold coverage value, that is, if there is a subset with k elements with a coverage value less or equal than the given threshold.

4.2.1. Set cover formulation

This reformulation of the feasibility problem for coverage representation is closely related to the approach described in Ponte et al. [16] for the ϵ-indicator, although it retains some of the characteristics of the threshold algorithm described in Section 3.2.

We reduce the feasibility problem to a k-set cover problem. For a given set of pairwise incomparable points $B = \{b_1, \ldots, b_n\}$, sorted with respect to the first component, and threshold coverage value c, let $B_1, \ldots, B_n$ be subsets of B, such that each subset B_i contains all elements b_j of B whose distance to b_i is smaller than c, i. e., $B_i = \{b_j \in B : ||b_i - b_j|| \leq c\}$. Then, the problem of finding a representative subset with k elements, whose coverage value is at most c, is equivalent to the k-set cover problem, i. e., finding k subsets such that the union of them covers every element of B. Note that, if less than k subsets already cover all elements in B, additional subsets can be chosen until k is reached since the coverage value does not increase by adding elements.

This set cover problem has a well-known integer linear programming formulation with a coefficient matrix A, where the rows $1, \ldots, n$ correspond to the indices of the n elements of B and the columns $1, \ldots, n$ correspond to the indices of the subsets $B_1, \ldots, B_n$. An element b_i is covered by the subset B_j if the entry $A_{i,j}$ is equal to one. The set cover problem consists of finding a minimum cardinality set of columns covering all rows of A. We now show that the matrix A of this particular set cover problem has the *consecutive ones property*, that is, the entries with a value of 1 in the matrix appear consecutively for each row.

Proposition 4.3. *For a given set $B = \{b_1, \ldots, b_n\} \subset \mathbb{R}^2$ of pairwise incomparable points, sorted with respect to their first component, (i.e., $b_1^1 < b_2^1 < \ldots < b_n^1$), and a threshold coverage value c, the rows of the coefficient matrix A of the k-set cover problem have the consecutive ones property (C1P).*

Proof. Let $b_i, x, y, z \in B \subset \mathbb{R}^2$ be pairwise incomparable points with $x^1 < y^1 < z^1$, and consider the ith row of A, i. e., the row corresponding to the subset $B_i = \{b_j \in B : ||b_i - b_j|| \leq c\}$. We show that if b_i covers the points x and z, then it also covers the point y. To this end, we assume that $||b - x|| \leq c$ and $||b - z|| \leq c$. Then we have to consider two cases: $b_i^1 < y^1$ and $b_i^1 > y^1$. If $b_i^1 < y^1$, then it follows directly from Proposition 2.1 that $||b_i - y|| < ||b_i - z|| \leq c$; if $b_i^1 > y^1$ we can conclude $||b_i - y|| < ||b_i - x|| \leq c$. In both cases, the distance from y to b_i is smaller than c. Thus, we can conclude that the rows of the covering matrix A satisfy the C1P. □

The algorithm presented in Ponte et al. [16] can be applied to the coverage representation problem, since all the required properties are met. It uses the algorithm proposed by Schöbel [18] to solve set cover problems with C1P. Moreover, the improvements presented by Vaz et al. [17] may also be applied, yielding a time complexity of $\mathcal{O}\left(n^2 \log n\right)$.

4.2.2. Shortest path formulation

The feasibility problem for coverage can also be reformulated as a shortest path problem. Although the previously presented approach is more efficient, it is harder to adapt to representation problems with more than one quality measure, as we will discuss in Section 6.

We define a directed graph $G_C = (N, E_C)$ with node set N and arc set E_C. The graph contains the following $n + 2$ nodes: N_s as a source node, N_t as a target node, and N_i as a node for each element

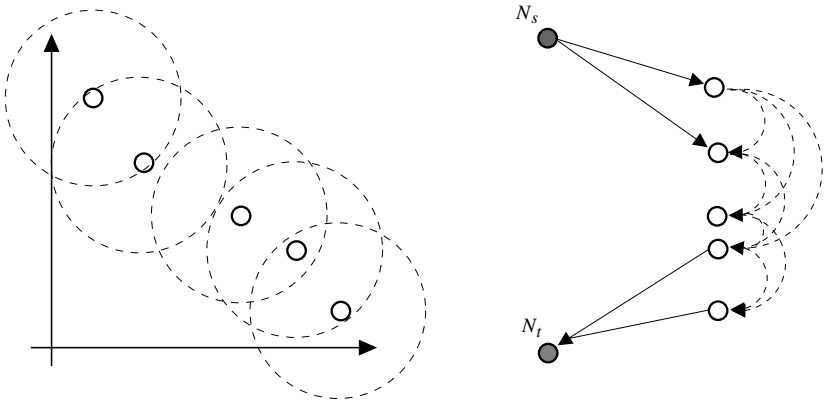

Figure 3: Threshold approach for coverage problem: set B with circles of radius c (left) and respective graph G_C (right)

$b_i \in B$, $i = 1, \ldots, n$. The graph G_C has an arc (N_s, N_i) if and only if $||b_i - b_1|| \leq c$. Similarly, there is an arc (N_i, N_t) if and only if $||b_i - b_n|| \leq c$. Let $s_i = \min\{j : ||b_i - b_j|| \leq c, j \in \{1, \ldots, n\}\}$ and $e_i = \max\{j : ||b_i - b_j|| \leq c, j \in \{1, \ldots, n\}\}$. Then, there is an arc (N_i, N_j) with $i < j$, if $s_j \leq e_i + 1$.

A path in G_C from N_s to N_t corresponds to a representative subset for a given threshold coverage value c. This holds since N_s is only connected to nodes that represent elements that cover the first element. Moreover, the structure of the graphs ensures that if there is an edge between two nodes N_i and N_j in G_C, then all elements between b_i and b_j are covered if the nodes b_i and b_j are selected, or, in other words, if the corresponding edge is selected in an N_s-N_t-path. Finally, only elements that cover the last element are connected to N_t, which means that every element, from b_1 to b_n is covered. Figure 3 shows an example of this transformation.

The shortest N_s-N_t-path in G_C corresponds to the smallest representative subset. To find a shortest path, a greedy strategy can be applied: If N_i is a node in a shortest path, then in at least one shortest path its successor $N_{j'}$ satisfies $j' = \max\{j : (N_i, N_j) \in E_C\}$. Since in every node of such a shortest path one has to look only for the largest possible step, the time complexity of this procedure is $\mathcal{O}(n)$. Note that alternatively, the algorithm of Dijkstra could be applied. If the shortest N_s-N_t-path uses at most $k+1$ arcs, then there is a feasible representative subset with a coverage radius of c or less, and that path represents a feasible representative subset with at most k elements. The time complexity of this procedure is $\mathcal{O}\left(n^2\right)$, since the algorithm needs to visit each arc to find a shortest path from N_s to N_t. Therefore, we now have $T_C(n) = \mathcal{O}(n)$ and an overall time complexity of $\mathcal{O}\left(n^2 \log n\right)$.

4.3. Experimental analysis

Two tests were performed in order to compare the four versions of the dynamic programming algorithm and the threshold algorithm (using the efficient set cover formulation) for the coverage representation problem with respect to the influence of n and k on the running time. As before, the implementations were run using randomly generated instances, each repeated five times.

For a given constant k, the results for the five approaches are shown in Figure 4, left, where DP1 corresponds to the dynamic programming approach described in Section 4.1, DP2 to the dynamic programming with the m-improvement described in Section 4.1.1, DP3 to the dynamic programming with the ℓ-improvement described in Section 4.1.2, DP4 to the dynamic programming with the improvement in Section 4.1.3, and TA to the threshold algorithm. The first dynamic programming DP1 approach is very slow in practice. This is expected given that it has $\mathcal{O}\left(k\, n^3\right)$ time complexity. Note that the algorithm takes more time to terminate for $n = 200$ than the remaining approaches for $n = 2000$. Regarding the remaining versions, the running times seem to be consistent with the theoretical time complexities. Perhaps the largest surprise is the small difference between the threshold algorithm and the DP2 and DP3 versions. Also interesting are the low running times of the DP4 approach, which clearly outperforms all other approaches.

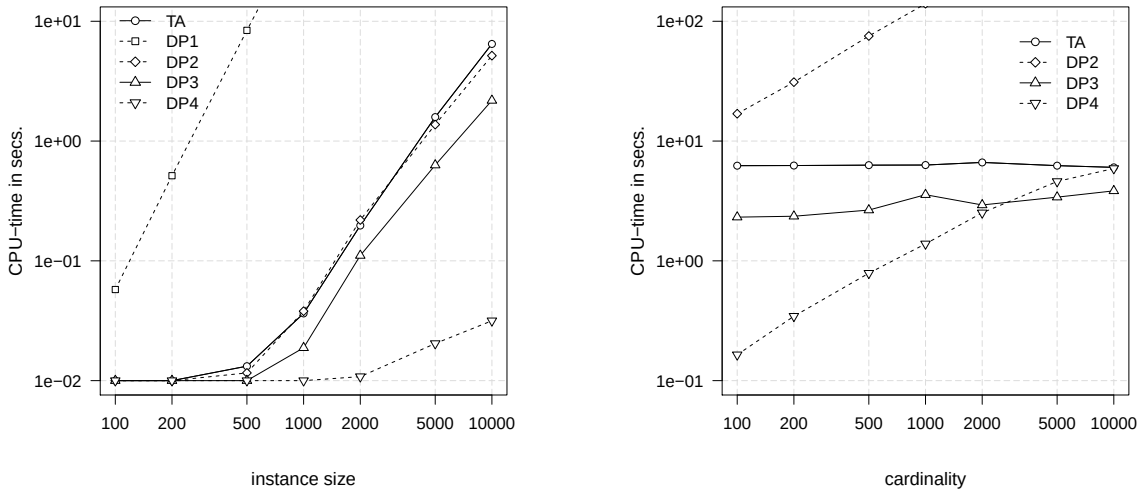

Figure 4: Running time of the dynamic programming and threshold approach for the coverage problem w. r. t. instance size n on the left (with $k = 20$ fixed) and cardinality k on the right side ($n = 10000$)

However, since the value for k is relatively small, and the complexity of this algorithm is $\mathcal{O}(k\,n)$, this is expected.

The right plot of Figure 4 shows the running time when n is kept constant. The first version DP1 does not even appear in the graph, since it takes too long to complete even the smaller case ($k = 50$, $n = 2000$), taking 1383 seconds. The relevance of the factor k in the complexity also becomes clear, since the version with time complexity $\mathcal{O}\left(k\,n^2\right)$ is affected by the increase in the value of k. In fact, if we consider $k = 500$, DP2 is around 10 times slower than the other versions. Additionally, while DP4 performs much faster than the other approaches for low values of k, the running times become more similar between DP3 and DP4 as k increases, and DP3 even outperforms DP4 for some values of k.

5. The ϵ-Indicator Representation Problem

Proposition 2.3 and Corollary 2.4 for the ϵ-indicator representation problem on the one hand, and Proposition 2.1 and Corollary 2.2 for the coverage representation problem on the other hand show the close relationship between the ϵ-indicator representation problem and the coverage representation problem. This justifies the application of slightly adapted versions of the algorithmic approaches described in Section 4 to the case of the ϵ-indicator representation problem. Since the numerical experiments did not provide additional information as compared to the coverage representation problem, they are omitted here.

5.1. Dynamic programming

The dynamic programming algorithm presented in Section 4.1 for the coverage indicator may be adapted for the ϵ-indicator, taking some caution with the way in which the values are calculated. As in Section 4.1 we denote by $S_{i,j}$ ($i \in \{1,\ldots,n\}$, $j \in \{1,\ldots,n-i+1\}$) the optimal representation of cardinality i of the set $\{b_j,\ldots,b_n\}$ that contains b_j and only elements taken from $\{b_j,\ldots,b_n\}$. Then, for $\ell > j$, the following holds:

$$I_\epsilon(\{b_j\} \cup S_{i-1,\ell}, \{b_j,\ldots,b_n\}) = \max\left\{\delta_{j,\ell}, I_\epsilon(S_{i-1,\ell}, \{b_\ell,\ldots,b_n\})\right\}$$

where

$$\delta_{j,\ell} = \max_{j \le m \le \ell} \min\left\{\epsilon(b_j, b_m), \epsilon(b_\ell, b_m)\right\}.$$

Therefore, the dynamic programming version uses an initial value of $T(1,j) = \epsilon(b_j, b_n)$ and the recursive formula

$$T(i,j) = \min_{j<\ell\leq n-i+2} \max\left\{\delta_{j,\ell}, T(i-1,\ell)\right\}.$$

As with the coverage indicator we need to apply a final correction to the result in order to consider the elements that are not included in the subproblems. Therefore, the final result is given by

$$\min_{1\leq j\leq n-k+1} \max\left\{\epsilon(b_j, b_1), T(k,j)\right\}.$$

Similarly to the algorithm for the coverage indicator, this approach has time complexity $\mathcal{O}(k\,n)$, using all the improvements mentioned before.

5.2. Threshold approach

The set covering procedure presented in Section 4.2.2 may also be adapted for the ϵ-indicator problem; see the discussion on related approaches in a different context in Ponte et al. [16] and Vaz et al. [17]. We simply need to replace the norm, as used in the coverage case, with the ϵ-indicator value, as defined in Section 2.4. However, some caution must be taken to ensure that the arguments of the ϵ-indicator function are provided in the right order. This is necessary because, unlike the norm in the coverage problem, the ratio in the ϵ-indicator is not commutative. Borrowing the notation of Section 4.2, every subset B_i "covered" by an element b_i corresponds to a column of the covering matrix A, hence, b_i is always the first argument of $\epsilon(\cdot,\cdot)$. Consequently, the potentially covered elements $b_1, \ldots, b_n$ of B correspond to the rows of the covering matrix A and thus to the second argument of $\epsilon(\cdot,\cdot)$. To conclude, we only need to prove that the consecutive ones property (C1P) holds for the columns of the covering matrix A when considering the ϵ-indicator problem.

Proposition 5.1. *Let $B = \{b_1, b_2, \ldots, b_n\} \subset \mathbb{R}^2$ be a set of pairwise incomparable points, sorted with respect to their first component (i.e., $b_1^1 < b_2^1 < \ldots < b_n^1$). For a given threshold value of the ϵ-indicator, t_ϵ, the columns of the covering matrix have the consecutive ones property (C1P).*

This result, as well as the analogous case for rows, is demonstrated by Vaz et al. [17].

The time complexity of $\mathcal{O}\left(n^2 \log n\right)$ also holds for this approach; see Section 4.2. However, we remark that a better time complexity may be achieved by the approach of Bringmann, Friedrich and Klitzke [19]. While the approach described above can be extended to multiobjective representation problems as described in the following sections, this remains an open question for the method of Bringmann, Friedrich and Klitzke.

Also note that, similar to Section 4.2.2, a shortest path formulation based on a corresponding graph $G_E = (N, E_E)$ (defined as the graph G_C in Section 4.2.2, using the ϵ-indicator instead of the norm to define the arc set E_E), can be given. We omit the details due to the similarity to Section 4.2.2.

6. Compromising between Quality Measures

While the two quality measures coverage and ϵ-indicator share some similarities in problem structure and often lead to similar (but in general not equal) representations in numerical experiments, each of which performs reasonably well also with respect to the other criterion, this is generally not the case for the uniformity criterion. A compromise between these quality measures can naturally be found in the efficient set of the multiobjective representation problems introduced in Section 2.5: The coverage–uniformity representation problem (CUR), the ϵ-indicator–uniformity representation problem (EUR), the coverage–ϵ-indicator representation problem (CER), and finally the three-objective representation problem (CEUR) considering all three of the above criteria. In the following we analyze the applicability of the solution approaches for the individual quality measures uniformity, coverage and ϵ-indicator to these multiobjective representation problems.

6.1. The Coverage – Uniformity Representation Problem

As the corresponding single-objective versions, the biobjective coverage–uniformity representation problem (CUR) can be solved by a dynamic programming algorithm and by a threshold approach, both of which are outlined in the following in relation to their single-objective counterparts.

6.1.1. Dynamic programming

The dynamic programming approaches described in the previous sections can be naturally extended to this biobjective representation problem by keeping, at each iteration, the non-dominated set of solutions for the corresponding subproblem. Formally, the base case is now $T(1,j) = (||b_j - b_n||,\ -\infty)$ and we have:

$$T(i,j) = \underset{j<\ell\leq n-i+2}{\text{vmin}} \left\{ \left(\max\left\{\delta_{j,\ell}, c\right\}, -\min\left\{||b_j - b_\ell||, -u\right\}\right) : (c,u) \in T(i-1,\ell) \right\},$$

where vmin represents the set of non-dominated vectors. The value $\delta_{j,\ell}$ is defined in Section 4.1.

The final correction described in Section 4.1 needs to be applied to the coverage component of every solution. Therefore, the final solutions are given by:

$$\underset{1\leq j\leq n-k+1}{\text{vmin}} \left\{ \left(\max\left\{||b_1 - b_j||, c\right\}, u\right) : (c,u) \in T(k,j) \right\}.$$

From these recursive equations it is clear that it is not possible to apply the improvement described in Sections 3.1 and 4.1, since choosing just one value of ℓ may yield the optimal solution for one of the indicators, but it may not give all of the required compromise solutions. Therefore, we do not use this improvement in any of the multiobjective dynamic programming variants.

Regarding the time complexity of this algorithm, we have to consider that the number of states of the dynamic programming algorithm is $\mathcal{O}(k\,n)$, i.e., the matrix T consists of $\mathcal{O}(k\,n)$ states representing sets of non-dominated objective vectors. For each of the states $T(i,j)$ of this matrix, the dynamic programming algorithm checks $\mathcal{O}(n)$ values. Each state may store at most $\mathcal{O}\left(n^2\right)$, since there are at most n^2 values for coverage, and for each value of coverage there is at most one non-dominated vector. Therefore, their lookup from one state and insertion in the other has time complexity $\mathcal{O}\left(n^2 \log n\right)$. This time complexity can be realized by using the technique described in Kung et al. [20] with an AVL binary balanced tree to keep the non-dominated vectors. Therefore, the final time complexity is $\mathcal{O}\left(k\,n^4 \log n\right)$.

6.1.2. Threshold approach

For the coverage–uniformity representation problem, the threshold algorithm must be adapted to handle the search in a bidimensional space and to keep a set of non-dominated objective vectors at each iteration. Let us denote the set of possible values for uniformity and coverage by $U = \{u_1, \ldots, u_{m_U}\}$ and $C = \{c_1, \ldots\ c_{m_C}\}$, respectively, with $m_U, m_C \leq n^2$. In addition, assume that $u_1 < \ldots < u_{m_U}$, and $c_1 < \ldots < c_{m_C}$.

The search method starts by considering the best possible value for one of the indicators, and the worst value for the other in any non-dominated objective vector. Since, in this case, we want to minimize coverage and maximize uniformity, we may start with the lowest coverage and uniformity values. Let j, ℓ be the indices of the considered values for uniformity and coverage, respectively. We choose $\ell = 1$, since c_1 is the best possible value for coverage, and $j = 1$, since u_1 is the worst value for uniformity.

In each iteration, the threshold values for both criteria are updated based on two basic operations: Either the first objective is worsened, i.e., the coverage is increased ($\ell \leftarrow \ell + 1$), or the second objective is improved which corresponds to increasing uniformity ($j \leftarrow j + 1$). For a given pair of threshold values, the algorithm determines if there is a subset such that its coverage and uniformity values do not exceed the threshold values. Formally, the goal is to find a representative subset $R \subseteq B$, $|R| = k$, such that $I_C(R,B) \leq c_\ell$ and $I_U(R) \leq u_j$, where c_ℓ and u_j are the current threshold values for coverage and uniformity, respectively.

Two cases need to be considered. If there is a feasible solution for the current threshold values, the uniformity value is increased, since increasing the value of coverage will only produce dominated objective vectors. If no feasible solution is found, then the coverage value is increased. Note that by increasing the uniformity value in this case, the set covering procedure would become more constrained and would not produce feasible solutions. By using this strategy, we need to check $\mathcal{O}(|U| + |C|) = \mathcal{O}\left(n^2\right)$ pairs of

threshold values to obtain all efficient solutions. Additionally, given that the algorithm visits the pairs of threshold values in increasing order of coverage, the procedure for filtering for non-dominated objective vectors becomes very simple.

Since the feasibility problem for uniformity and for coverage can both be solved using the correspondence of representative subsets and paths in the corresponding graph G_C and G_U, respectively (shortest and longest paths, respectively, see Section 3.2 for uniformity and Section 4.2.2 for coverage), we use a combination of the two approaches to solve the coverage–uniformity representation problem.

In either of the individual threshold algorithms, each representative subset is represented by a path in the respective graph. Therefore, we want to find a path that is valid in both graphs, that is, a representative subset that is a feasible solution according to both indicators and the respective thresholds. Since we want to find a path that exists in both graphs G_C and G_U, and since the graphs have the same set of vertices N, we can intersect the respective sets of arcs E_C and E_U, which results in a new graph denoted by $G_{C,U}$. Finding a path in this graph is equivalent to finding a path that exists in both graphs, since only arcs that exist in both graphs exist in $G_{C,U}$.

However, uniformity and coverage are different in the sense that adding elements to a subset may decrease the uniformity, while removing elements may increase the coverage. Therefore, for a subset to be a feasible solution of the problem, we must find a path with exactly $k+2$ nodes, including the nodes N_s and N_t. This search can be implemented as follows: The nodes of $G_{C,U}$ are visited in the order $N_s, N_1, \ldots, N_n, N_t$. The possible lengths of paths from N_s to the current node N_i are stored at this node. Then, while considering the node N_i, all outgoing arcs are visited and the lengths stored in the adjacent nodes are updated. If $k+1$ is a possible path length from N_s to N_t, then there is a feasible solution which is represented by the corresponding path. Note that we only need to store the minimum and maximum possible path lengths at each node since every path length in that interval is possible, see Proposition 6.1.

Proposition 6.1. *Given a graph $G_{C,U}$, and two $s-t-$paths, P and Q, represented as sequences of nodes, then for each ℓ, $|P| < \ell < |Q|$, there is a path P', such that $|P'| = \ell$.*

Proof. It is sufficient to show that for two paths P and Q with $|P| \leq |Q| - 2$ a path P' exists with $|P| < |P'| < |Q|$. Assume that the two paths P and Q differ between the nodes $\tilde{s}$ and $\tilde{t}$, and that the sub-paths from $\tilde{s}$ to $\tilde{t}$ in P and Q, denoted by $P|_{[\tilde{s},\tilde{t}]}$ and $Q|_{[\tilde{s},\tilde{t}]}$, respectively, are node (and edge) disjoint and have a length difference of at least two, i.e.,

$$\left|P|_{[\tilde{s},\tilde{t}]}\right| \leq \left|Q|_{[\tilde{s},\tilde{t}]}\right| - 2.$$

Note that if such subpaths would not exist, then there would be at least two parts in P and Q in which the paths are disjoint, namely on the intervals $[s^1, t^1]$ and $[s^2, t^2]$, and the length difference of the respective subpaths would differ only by one: $\left|Q|_{[s^1,t^1]}\right| - \left|P|_{[s^1,t^1]}\right| = 1$ and $\left|Q|_{[s^2,t^2]}\right| - \left|P|_{[s^2,t^2]}\right| = 1$. In this case one would be able to construct a path $P' = P|_{[s,t^1]} \cup Q|_{[t^1,t]}$, for which $|P| < |P'| < |Q|$ holds. Thus, we can restrict ourselves to the above mentioned case, i.e., the two subpaths $P|_{[\tilde{s},\tilde{t}]}$ and $Q|_{[\tilde{s},\tilde{t}]}$ are node disjoint and their lengths differ by at least two.

Then, necessarily there are two consecutive nodes of the longer path $Q|_{[\tilde{s},\tilde{t}]}$ without a node of $P|_{[\tilde{s},\tilde{t}]}$ (the shorter path) in between. Thus, we have:

$$Q|_{[\tilde{s},\tilde{t}]} = \tilde{s}, \ldots, i-\ell, i, j, j+m, \ldots, \tilde{t}$$
$$P|_{[\tilde{s},\tilde{t}]} = \tilde{s}, \ldots, i-q, j+p, \ldots, \tilde{t}$$

with $\ell, m, p, q \geq 1$. Moreover, a pair of nodes with this property can be selected such that $|P|_{[\tilde{s},i-q]}| = |Q|_{[\tilde{s},i-\ell]}|$.

Since $Q|_{[\tilde{s},\tilde{t}]}$ and $P|_{[\tilde{s},\tilde{t}]}$ are feasible paths representing solutions of the feasibility problem for uniformity and for coverage, we can conclude that the edges $(i, j+p)$ and $(i-q, j)$ are also feasible. Therefore, the path $P' = P|_{[\tilde{s},i-q]} \cup (i-q, j) \cup Q|_{[j,\tilde{t}]}$ is also feasible and has length $|Q| - 1$.

□

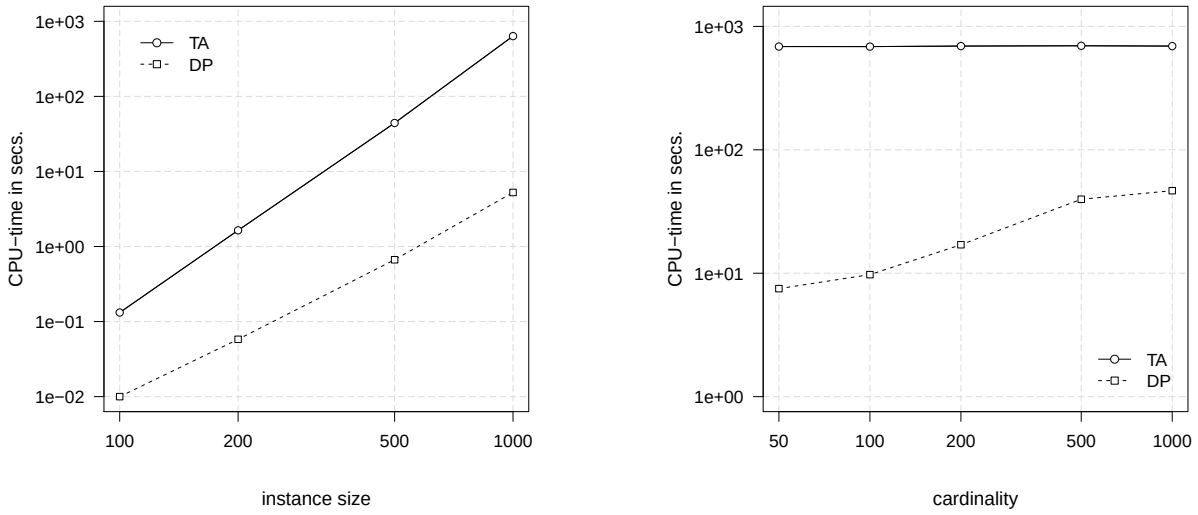

Figure 5: Running time of the dynamic programming and threshold approach for the coverage–uniformity representation problem w. r. t. instance size n on the left (with $k = 20$ fixed) and cardinality k on the right ($n = 1000$)

We can conclude that the threshold algorithm visits $\mathcal{O}\left(n^2\right)$ pairs of threshold values. Therefore, using the notation defined in Section 3.2, we have $T_S(n) = \mathcal{O}\left(n^2\right)$. For each pair of threshold values, the feasibility problem implies the following two essential tasks and time complexities:

- Generate the graph $G_{C,U}$ which can be done in $\mathcal{O}\left(n^2\right)$ by checking every pair of elements;
- Find the lengths of the feasible paths in the graph $G_{C,U}$, which can be done in $\mathcal{O}\left(n^2\right)$ since we visit each node and each arc once.

In conclusion, the feasibility problem has a time complexity of $T_C(n) = \mathcal{O}\left(n^2\right)$, and therefore the overall time complexity for the threshold algorithm is $\mathcal{O}\left(n^4\right)$.

6.1.3. Experimental analysis

In order to compare the performance of the two algorithms, we ran two experiments to test the influence of the parameters n and k on the running time. As performed in Section 3.3, we use five runs on each instance, with randomly generated instances.

In Figure 5, the results for the first test are presented. Similar to the results for the previous problems, the results indicate that the dynamic programming version performs much better than the threshold algorithm.

These results are not consistent with the time complexity, since we would expect that the algorithm with lower complexity would perform better. However, the difference in time complexities is only $\mathcal{O}\left(k \log n\right)$, and in this test k is constant. Moreover, the time complexity for the dynamic programming version is an upper bound for the worst case scenario, considering the maximum number of non-dominated vectors stored in each entry. If the size of the set is lower than the maximum value, then that will reflect on the running time, explaining the difference in performances of the algorithms.

The second test, on the other hand, does not consider k as a constant value, so that we could expect different results. The performances for the two algorithms, with a fixed value of n and varying k, are presented in Figure 5. However, even though the performance of the dynamic programming algorithm decreases with k, it still has much better performance than the threshold algorithm. Moreover, although the dynamic programming algorithm uses more memory due to the fact that it needs to store the non-dominated objective vectors for every subproblem, this does not affect its performance. In conclusion, this version of

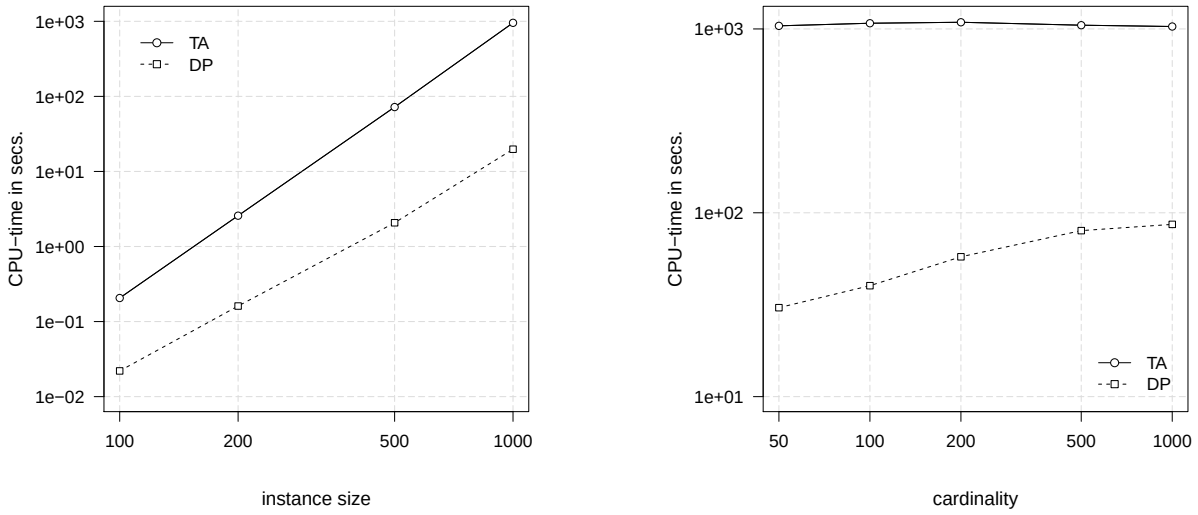

Figure 6: Running time of the dynamic programming and threshold approach for the ϵ-indicator–uniformity representation problem w. r. t. instance size n on the left (with $k = 20$ fixed) and cardinality k on the right ($n = 1000$)

the dynamic programming algorithm has better performance than the threshold algorithm, independent of the value of k.

6.2. The ϵ-Indicator – Uniformity Representation Problem

Due to the close relationship between the coverage indicator and the ϵ-indicator as discussed already in Section 5, the methods derived for the coverage–uniformity representation problem can be easily adapted to the case of the ϵ-indicator–uniformity representation problem (EUR). In fact, if pairwise distances and ϵ-indicator values are computed in a preprocessing step, the same methods can be applied using the respective distance matrices. Thus, all results described in the previous section transfer to this case.

The experimental results for this problem, presented in Figure 6, confirm that the problems are similar, even though the approaches for the coverage–uniformity representation problem are faster by a small constant factor. This confirms that the indicators, while similar in structure, may generate different representations, and different numbers of non-dominated vectors, which may influence the running time.

6.3. The Coverage – ϵ-Indicator Representation Problem

6.3.1. Dynamic programming

The dynamic programming algorithm for the coverage–ϵ-indicator representation problem (CER) is adapted from the single objective versions from Sections 4.1 and 5.1, as done for the coverage–uniformity representation problem, see Section 6.1.1.

The new base case is given by $T(1, j) = (||b_j - b_n||,\ \epsilon(b_j, b_n))$ and we have

$$T(i, j) = \operatorname*{vmin}_{j<\ell\leq n-i+2} \left\{ \left(\max\left\{\delta^C_{j,\ell}, c\right\}, \max\left\{\delta^\epsilon_{j,\ell}, e\right\}\right) : (c, e) \in T(i-1, \ell) \right\},$$

where vmin represents the set of non-dominated vectors and the values $\delta^C_{j,\ell}$ and $\delta^\epsilon_{j,\ell}$ are used as defined in Sections 4.1 and 5.1, respectively. The final solutions for the problem are given by

$$\operatorname*{vmin}_{1\leq j\leq n-k+1} \left\{ \left(\max\left\{||b_1 - b_j||, c\right\}, \max\left\{\epsilon(b_j, b_1), e\right\}\right) : (c, e) \in T(k, j) \right\}.$$

As in Section 6.1.1, we also use the technique of Kung et al. [20] and an AVL tree to store the solutions found for each subproblem. The time complexity of this algorithm is $\mathcal{O}\left(kn^4 \log n\right)$, similar to the algorithm for the combination of coverage and uniformity.

6.3.2. Threshold approach

In order to solve problem (CER) with the threshold approach, we use the same structure as in Section 6.1.2. In particular, we use a similar search method, adapted for coverage and ϵ-indicator. Since our goal is to minimize both indicators, the algorithm starts with the lowest possible value for coverage, and hence the highest value for the ϵ-indicator among the efficient solutions (in fact, we may even overestimate this value since we do not perform a lexicographic optimization). When a feasible solution is found for a given pair of threshold values, the threshold value for the ϵ-indicator is decreased. Otherwise, the threshold value for coverage is increased.

Regarding the feasibility problem that needs to be solved for a given pair of threshold values, one possible solution approach is to use a similar strategy as in Section 6.1.2 based on a combination of the graphs G_C and G_E. However, in this case an efficient alternative is to adapt the single-objective set cover algorithms using the consecutive ones property (see Sections 4.2.1 and 5.2) to consider both indicators simultaneously.

A first possible realization of this approach is to consider, instead of the $n \times n$ covering matrix for a single objective, a $2n \times n$ covering matrix constructed by appending the matrices for both indicators. In order to apply the same algorithm to this new matrix, the rows of this extended covering matrix would first need to be sorted lexicographically. Then we could proceed according to the operations described in Schöbel [18].

An alternative realization of the set cover approach is based on a modification of the single-objective algorithm so that it uses the covering matrices for both indicators for the selection of columns, while keeping the same idea. We start as in the single-objective version by picking the rightmost column such that the corresponding entry on the first row of both matrices has a value of 1. Then, as in the original algorithm, we skip the rows that have a value of 1 in that column, independently for each of the matrices. Since the matrices are in general different, the algorithm may, at that moment, point to different rows for each of the two matrices. In each of these rows, the rightmost column with a value of 1 is identified, and the column with the smaller index is selected. Then the process is repeated, along with skipping rows as described, until all rows in both matrices are considered. Since this algorithm still finds the solution in linear time, the complexity of this procedure is still $\mathcal{O}(n \log n)$, considering the overhead caused by the generation of the compressed covering matrix (see Vaz et al. [17]).

This second alternative can in fact be interpreted as a compressed implementation of the first alternative (where we assume that the first alternative is applied to the complete, lexicographically sorted, extended $2n \times n$ covering matrix). More precisely, if in some iteration a column j is selected, then the first row found in the complete, lexicographically sorted, extended $2n \times n$ matrix (i.e., the row found after skipping all rows with a value of 1 in column j) corresponds exactly to the row that is selected in the second alternative, since this row is lexicographically smaller (higher) than any other row for which the rightmost column with a value of 1 has a higher index.

Regarding the overall time complexity, the only part that is significantly altered in comparison to Section 6.1.2 is the feasibility check. Therefore, this algorithm has $T_S(n) = \mathcal{O}\left(n^2\right)$ and $T_P(n) = \mathcal{O}\left(n^2 \log n\right)$, which is the time complexity of sorting the list of values for both indicators. The feasibility check for this problem is more efficient than for the combination of coverage and uniformity, with a time complexity of $T_C(n) = \mathcal{O}(n \log n)$. Therefore, the final time complexity for this algorithm is $\mathcal{O}\left(n^3 \log n\right)$.

6.3.3. Experimental analysis

We now present the experimental results of the two approaches for this biobjective problem. As in previous sections, we performed two tests in order to evaluate the influence of both n and k on the running time. We chose to use values of $n \leq 1000$ for the first test and $k \leq 1000$ for the second. Figure 7 presents the results obtained.

The results for the first test indicate that the dynamic programming version performs better than the threshold algorithm by an order of magnitude. Given that the time complexity for the threshold algorithm is lower, the opposite would have been expected. However, as previously mentioned, the time complexities represent a worst case analysis, and in the case of dynamic programming there may be a significant difference between the worst case bound and the practical performance.

The second test, with the goal of studying the influence of k on the running time, shows that the dynamic programming approach, while faster for small values of k, is outperformed for $k \geq 500$. This increase in the

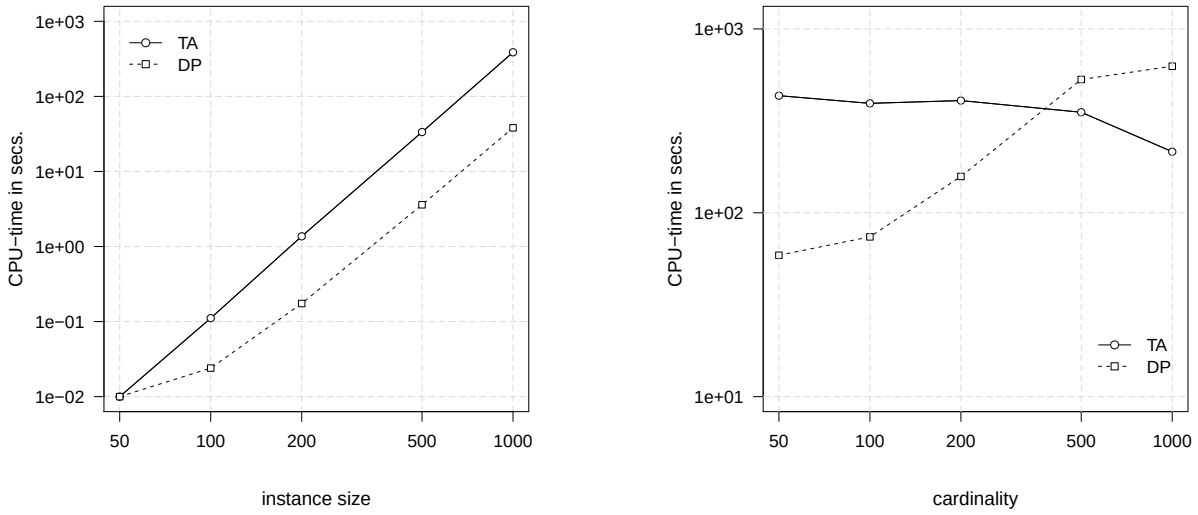

Figure 7: Running time of the dynamic programming and threshold approach for the coverage – ϵ-indicator problem w. r. t. instance size n on the left (with $k = 20$ fixed) and cardinality k on the right side ($n = 1000$)

running time is consistent with the time complexity of the dynamic programming algorithm, which grows linearly with k.

In conclusion, the dynamic programming approach clearly performs better than the threshold algorithm for relatively small values of k. However, while the threshold algorithm only uses memory to store the values of the indicators to test, and the necessary space to keep the final solutions, the dynamic programming approach uses much more memory which makes it impractical for large values of k.

6.4. The Coverage – ϵ-Indicator – Uniformity Representation Problem

6.4.1. Dynamic Programming

The dynamic programming algorithm for this problem is based on the biobjective version described in Section 6.1.1. In order to work with three objectives, we must, first of all, modify the algorithm so that it calculates and stores the vectors with the three objectives. Therefore, we now have $T(1, j) = (||b_j - b_n||, \epsilon(b_j, b_n), -\infty)$ and

$$T(i,j) = \operatorname*{vmin}_{j<\ell\leq n-i+2} \left\{ \left(\max\left\{\delta^C_{j,\ell}, c\right\}, \max\left\{\delta^\epsilon_{j,\ell}, e\right\}, -\min\left\{||b_j - b_\ell||, -u\right\}\right) : (c,e,u) \in T(i-1,\ell) \right\}$$

with $\delta^C_{j,\ell}$ and $\delta^\epsilon_{j,\ell}$ as defined in Sections 4.1 and 5.1. Additionally, the final solutions are generated with

$$\operatorname*{vmin}_{1\leq j\leq n-k+1} \left\{ \left(\max\left\{||b_1 - b_j||, c\right\}, \max\left\{\epsilon(b_j, b_1), e\right\}, u\right) : (c,e,u) \in T(k,j) \right\}.$$

The technique of Kung et al. [20] for the three-dimensional case, an extension of the technique described in Section 6.1.1, can be used to maintain a set of non-dominated vectors in linearithmic time complexity. We recall that there are at most n^2 values for each indicator. If we fix the values for two of the indicators, then there is at most one non-dominated vector. Since we have $\mathcal{O}\left(n^4\right)$ ways of choosing values for two of the indicators, the we have at most $\mathcal{O}\left(n^4\right)$ solutions for each subproblem. Therefore, sorting and removing the dominated objective vectors has $\mathcal{O}\left(n^4 \log n\right)$ time complexity. This is repeated for $k\,n$ entries of the matrix, and for each of them at most $\mathcal{O}\left(n\right)$ problems need to be checked. Therefore, the time complexity of the algorithm is bounded by $\mathcal{O}\left(k\,n^6 \log n\right)$. As with the biobjective problem, this is a worst case analysis which may not reflect on the running time observed in practice.

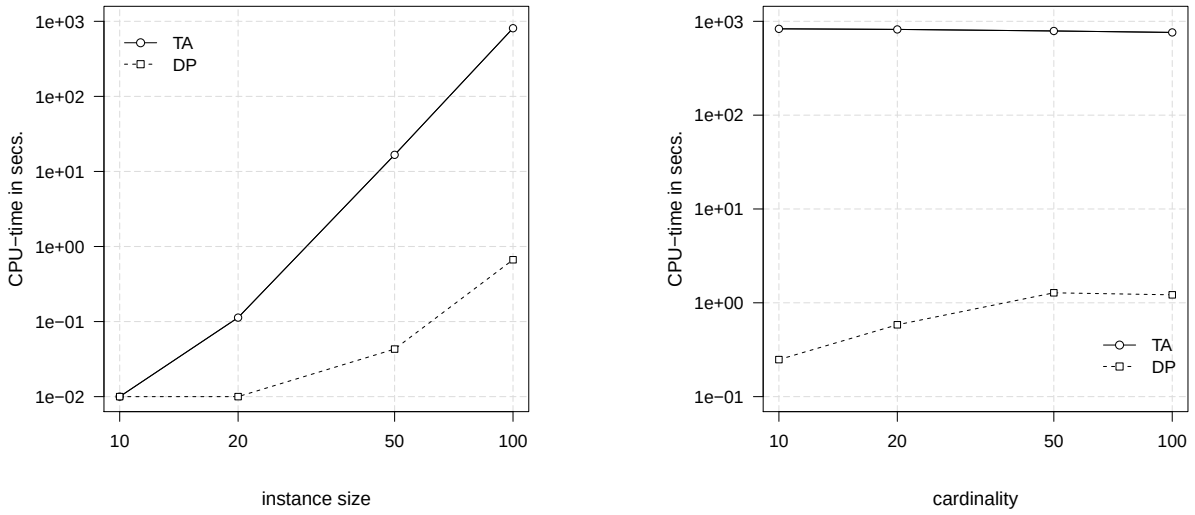

Figure 8: Running time of the dynamic programming and threshold approach for the coverage – ϵ-indicator – uniformity representation problem w. r. t. instance size n on the left (with $k = 20$ fixed) and cardinality k on the right side ($n = 100$)

6.4.2. Threshold approach

As with the biobjective problem described in Section 6.1.2, we must adapt the search method, as well as the set covering procedure, in order to use the threshold approach. For the search method, we chose to apply the two dimensional version, by fixing one of the indicators. In other words, for each value of coverage, the algorithm executes the search over the values of ϵ-indicator and uniformity, similarly to the method described in Section 6.1.2.

For the set covering procedure, we adapted the version described in Section 6.1.2. This time, we add the ϵ-indicator, and therefore add a new graph, G_E, which is analogous to the graph G_C as described in Section 5.2, see also Section 5.2. The graph used for the algorithm, $G_{C,E,U}$ is the intersection of the graphs G_C, G_E, G_U.

The proof of the correctness of the algorithm for coverage and uniformity is also valid when adding the ϵ-indicator, since the proof does not use any specific knowledge about underlying distances other than the covering matrix having the consecutive ones property. However, we have already seen that the ϵ-indicator has many of the properties of coverage, including C1P for the rows [17] and columns (Proposition 5.1). Even though we have one more graph to intersect (which produces some overhead), this does not increase the time complexity, and therefore the complexity for this procedure is $\mathcal{O}\left(n^2\right)$.

Finally, we discuss the method to store and filter the non-dominated objective vectors. Despite being relatively simple when the search space is two dimensional, when we add a third dimension, there is no natural order in which we can store the objective vectors, allowing us to quickly check if a vector is dominated. A possible solution to this problem is to use an AVL tree as described by Kung et al. [20], which results in a similar approach to that used for the dynamic programming algorithm. In particular, our implementation uses an AVL tree to store the 2D vectors and an ordinary array to store the 3D vectors. Moreover, the implementation tests each vector as soon as it finds it, so as to not waste memory with dominated vectors.

Given that we have, at most, n^4 objective vectors (one for each pair of ϵ-indicator and uniformity), then we must add $\mathcal{O}\left(n^4 \log n^4\right) = \mathcal{O}\left(n^4 \log n\right)$ to the complexity. Given that the search method has a complexity of $T_S(n) = \mathcal{O}\left(n^4\right)$, and the set covering procedure of $T_P(n) = \mathcal{O}\left(n^2\right)$, we have a final complexity of $\mathcal{O}\left(n^6\right)$ even after adding the extra complexity of $\mathcal{O}\left(n^4 \log n\right)$ for keeping only the non-dominated solutions.

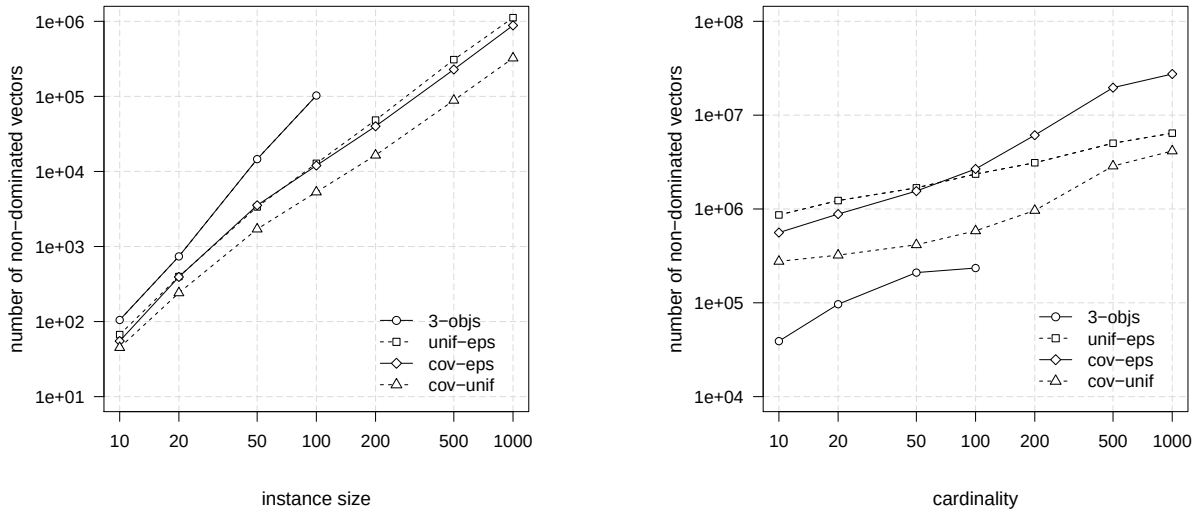

Figure 9: Number of non-dominated vectors stored by dynamic programming approaches for all multiobjective representation problem w. r. t. instance size n on the left (with $k = 20$ fixed) and cardinality k on the right side ($n = 100$ for "3-objs", $n = 1000$ for all others)

6.4.3. Experimental analysis

In this section, we present the experimental results for the coverage – ϵ-indicator –uniformity representation problem. As in the previous sections, we consider two different tests in order to measure the influence of n and k on the running time. However, as the algorithms have much larger complexities, we use lower limits for n and k.

The complexities of these algorithms are in $\Omega\left(n^6\right)$, so a small value such as $n = 100$ is already a challenge for current computers. In Figure 8, the results for the first test are presented. Since the algorithms have very different running times, we chose to represent the data using a logarithmic scale. It is clear that the dynamic programming approach performs much better, reaching a difference of 10^3 around $n = 70$. As mentioned in the previous sections, the complexity of the dynamic programming version is based on a worst case analysis that does generally not reflect its practical performance.

The second test, with the goal of studying the influence of k on the running time, confirms the superiority of the dynamic programming version. Although the dynamic programming version shows an increase in running time for larger values of k, it still outperforms the threshold algorithm by a large margin. This increase is consistent with the complexity of the dynamic programming algorithm, as is the absence of influence to the threshold algorithm.

6.5. Number of non-dominated vectors

In Figure 9, the total number of non-dominated vectors stored in the dynamic programming table for the various problems is presented, where `unif-eps`, `cov-eps` and `cov-eps` correspond to the three biobjective problems and `3-objs` to the problem with three objectives. These values were obtained simultaneously as the running times for the previously described experimental results, that is, the experimental setting is the same as described in the respective sections.

The results in the left plot suggest that the number of stored vectors is $\Theta\left(n^2\right)$ for combinations of two indicators and $\Theta\left(n^3\right)$ for the combination of all three indicators. Since the number of entries in the dynamic programming table is $\Theta\left(n\right)$ (for constant k), this suggests that the average number of vectors per entry is $\Theta\left(n\right)$ and $\Theta\left(n^2\right)$, for the two and three indicator versions, respectively. While this number quickly grows large, it is still short of our bounds of $\mathcal{O}\left(n^2\right)$ and $\mathcal{O}\left(n^4\right)$, which presents some evidence that there may

be some better estimations of the number of stored vectors, at least for the average case. We also remark that, while the number of returned non-dominated vectors is of practical importance, the total number of non-dominated vectors stored in the dynamic programming table greatly influences the runtime of the algorithm.

The results in the right plot are also interesting, since they suggest a sublinear dependence of the number of vectors on the value of k. Since the number of entries in the dynamic programming table is $\mathcal{O}(k\,n)$, it would be expected that the total number of vectors in the table would increase linearly with k. The results suggest that, for the rows of the table corresponding to larger values of k, the number of vectors may decrease, hence the sublinear dependence.

The exception is the algorithm for `cov-eps`, which produces a large quantity of solutions, if k is big enough. In fact, for $k \geq 500$, the number of intermediate solutions is too large to keep in memory in our setup, and the algorithm required an adjustment to release the memory associated with intermediate solutions as they were used. As a consequence, the algorithm simply outputs the values of coverage and ϵ-indicator, since the chosen subset is usually reconstructed using the intermediate solutions. This is not a problem, since the subset could easily be reconstructed using the techniques used for the threshold approach, but is evidence of the problems with the dynamic programming approach.

7. Discussion and Conclusions

In this work, we have formulated several representation problems recast as single and multiobjective optimization problems, as well as polynomial time algorithms based on the principles of dynamic programming and threshold approaches. An extensive experimental analysis on a wide range of randomly generated instances was conducted, which gave further insight into the conditions under which each approach performs the best.

Although having a larger worst-case time complexity on multiobjective representation problems, the dynamic programming approach performed faster, in practice, than the threshold algorithm. Tighter time-complexity bounds for dynamic programming should be obtainable, for instance, by using an average-case or smooth analysis argument on the number of states that are kept at each iteration; see, for example, Beier and Vöcking [21]. We recall that we have assumed the worst-case scenario at each iteration, which is very far from what we have observed in practice.

Besides being widely used in the context of multiobjective mathematical programming, the representation problem also arises in subset-selection procedures and archiving in heuristic approaches, see e.g. [22, 16]; at each iteration, a subset of a pool of candidate solutions that promotes some particular notion of representation quality is chosen. For this reason, procedures that are efficient in practice are strongly required. The competitive running times of the dynamic programming algorithms indicate that they are very suitable for efficient subset-selection procedures and archiving.

Unfortunately, finding representations of the non-dominated set for more than two objectives may become an intractable task. The algorithms proposed in this article cannot be easily modified for such cases. For instance, the consecutive ones property does not hold in general in the set covering problem, see [17]. Moreover, the shortest path formulation is not valid anymore for more than two objectives. Note that the related k-dispersion and k-center problems are NP-hard in general. Therefore, the development of approximation algorithms for these problems is of particular interest.

Finally, we remark that little is known about the relation between these indicators. Further advances on this topic would be particularly useful since some indicators may give similar outcomes under certain conditions. The multiobjective formulation of the representation problem that is explored in this article provides the basis for studying the relation between indicators in an experimental setting.

Acknowledgments. The authors wish to acknowledge the anonymous reviewers for the detailed and helpful comments to the manuscript. This work was supported by the bilateral cooperation project "RepSys - Representation systems with quality guarantees for multi-objective optimization problem" funded by the Deutscher Akademischer Austausch Dienst and Conselho de Reitores das Universidades Portuguesas and partially supported by iCIS (CENTRO-07-ST24-FEDER-002003).

References

[1] H. Benson and S. Sayin, Towards finding global representations of the efficient set in multiple objective mathematical programming, Naval Research Logistics 44 (1997) 47–67.

[2] S. Sayin, Measuring the quality of discrete representations of efficient sets in multiple objective mathematical programming, Mathematical Programming 87 (3) (2000) 543–560.

[3] S. Ruzika and M. Wiecek, A survey of approximation methods in multiobjective programming, Journal of Optimization Theory and Applications 126 (3) (2005) 473–501.

[4] J. Sylva and A. Crema, A method for finding well-dispersed subsets of non-dominated vectors for multiple objective mixed integer linear programs, European Journal of Operational Research 180 (2007) 1011–1027.

[5] S.L. Faulkenberg and M. Wiecek, On the quality of discrete representations in multiple objective programming, Optimization and Engineering 11 (2010) 423–440.

[6] A. Eusebio, J.R. Figueira and M. Ehrgott, On finding representative non-dominated points for bi-objective integer network flow problems , Computers and Operations Research 48 (2014) 1–10.

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

[8] J. Edmonds, D. R. Fulkerson, Bottleneck extrema, Journal of Combinatorial Theory 8 (1970) 299–306.

[9] R. Steuer, Multiple Criteria Optimization: Theory, Computation and Application, John Wiley, New York, 1986.

[10] S. Ravi, D. Rosenkrantz, G. Tayi, Facility dispersion problems: Heuristics and special cases, in: F. Dehne, J.-R. Sack, N. Santoro (Eds.), Algorithms and Data Structures, Vol. 519 of Lecture Notes in Computer Science, Springer, 1991, pp. 355–366.

[11] O. Kariv, S. Hakimi, An algorithmic approach to network location problems i: The p-centers, SIAM Journal on Applied Mathematics 37 (1979) 513–538.

[12] R. Hassin, A. Tamir, Improved complexity bounds for location problems on the real line, Oper. Res. Lett. 10 (7) (1991) 395–402.

[13] A. Schöbel, Locating stops along bus or railway lines - a bicriteria problem, Annals of Operations Research 136 (2005) 211–227.

[14] C. H. Papadimitriou, M. Yannakakis, On the approximability of trade-offs and optimal access of web sources, in: Proceedings of the 41st Annual Symposium on Foundations of Computer Science, FOCS '00, IEEE Computer Society, Washington, DC, USA, 2000, pp. 86–92.

[15] D. W. Wang, Y.-S. Kuo, A study on two geometric location problems, Information Processing Letters 28 (6) (1988) 281–286.

[16] A. Ponte, L. Paquete, J. Figueira, On beam search for multicriteria combinatorial optimization problems, in: N. Beldiceanu, N. Jussien, E. Pinson (Eds.), Integration of AI and OR Techniques in Contraint Programming for Combinatorial Optimzation Problems - 9th International Conference, CPAIOR 2012, Nantes, France, May 28 - June 1, 2012, Proceedings, Vol. 7298 of Lecture Notes in Computer Science, Springer, 2012, pp. 307–321.

[17] D. Vaz, L. Paquete, A. Ponte, A note on the ϵ-indicator subset selection, Theoretical Computer Science 499 (2013) 113 – 116.

[18] A. Schöbel, Set covering problems with consecutive ones property, Tech. Rep. 2005-03, Georg-August Universität Göttingen, Institut für Numerische und Angewandte Mathematik (2005).

[19] K. Bringmann, T. Friedrich and Patrick Klitzke, Two-dimensional subset selection for hypervolume and epsilon-indicator, in: Proceedings of the 16th Annual Conference on Genetic and Evolutionary Computation, GECCO 2014, ACM Press, New York, NY, USA, 2014, pp. 589–596.

[20] H. T. Kung, F. Luccio, F. P. Preparata, On finding the maxima of a set of vectors, Journal of the Association for Computing Machinery 222 (4) (1975) 469–475.

[21] R. Beier, B. Vöcking, Random knapsack in expected polynomial time, in: L. Larmore, M. Goemans (Eds.), Proc. of the 35rd Annual ACM Symposium on Theory of Computing (STOC), ACM Press, 2003, pp. 232–241.

[22] J. Bader, Hypervolume-Based Search for Multiobjective Optimization: Theory and Methods, Ph.D. thesis, ETH Zurich, Switzerland (2010).