Mathematical Methods of Operations Research
https://doi.org/10.1007/s00186-023-00841-0

ORIGINAL ARTICLE

A simple, efficient and versatile objective space algorithm for multiobjective integer programming

Kerstin Dächert[1] · Tino Fleuren[2] · Kathrin Klamroth[3]

Received: 26 February 2023 / Revised: 2 August 2023 / Accepted: 31 October 2023

Abstract
In the last years a multitude of algorithms have been proposed to solve multiobjective integer programming problems. However, only few authors offer open-source implementations. On the other hand, new methods are typically compared to code that is publicly available, even if this code is known to be outperformed. In this paper, we aim to overcome this problem by proposing a new state-of-the-art algorithm with an open-source implementation in C++. The underlying method falls into the class of objective space methods, i.e., it decomposes the overall problem into a series of scalarized subproblems that can be solved with efficient single-objective IP-solvers. It keeps the number of required subproblems small by avoiding redundancies, and it can be combined with different scalarizations that all lead to comparably simple subproblems. Our algorithm bases on previous results but combines them in a new way. Numerical experiments with up to ten objectives validate that the method is efficient and that it scales well to higher dimensional problems.

Keywords Multiobjective optimization · Nondominated set · Objective space algorithms · Search region · Local upper bounds · Scalarization

Tino Fleuren and Kathrin Klamroth have been contributed equally to this work.

✉ Kerstin Dächert
kerstin.daechert@htw-dresden.de

Tino Fleuren
tino.fleuren@itwm.fraunhofer.de

Kathrin Klamroth
klamroth@uni-wuppertal.de

1 Faculty of Informatics/Mathematics, Hochschule für Technik und Wirtschaft Dresden - University of Applied Sciences, Friedrich-List-Platz 1, 01069 Dresden, Germany

2 Fraunhofer Institute for Industrial Mathematics, Fraunhofer-Platz 1, 67663 Kaiserslautern, Germany

3 Department of Mathematics and Computer Science, University of Wuppertal, Gaußstraße 20, 42119 Wuppertal, Germany

Published online: 21 March 2024

1 Introduction

Integer programming (IP) is a classical field of Operations Research with a wide range of economical and industrial applications. Prominent examples are knapsack and capital budgeting problems, assignment problems, routing problems and problems occuring in supply chain management applications. The practical success of integer programming is supported by the fact that highly efficient solvers are readily available. This is not the case for multiobjective integer programming (MOIP). Indeed, while there is a growing need for the consideration of multiple conflicting goals, including economical, ecological and robustness criteria, the development of MOIP solvers lags behind.

A simple yet efficient way to overcome this shortcoming can be seen in the recent trend towards so-called *objective space methods* or *scalarization-based algorithms*. Different from "purely" multiobjective approaches like, for example, multiobjective branch and bound algorithms, such methods rely on the iterative solution of appropriately defined single-objective IPs, hence taking advantage of the efficiency of single-objective IP solvers. Besides the fact that such methods largely benefit from the strength of single-objective IP solvers, they are independent of the particular problem structure and are thus very generally applicable, without the hassle of problem specific fine-tuning.

Objective space methods can be distinguished w.r.t. the way the objective space is decomposed and w.r.t. the way the subproblems are formulated. Both aspects, *decomposition* and *subproblem formulation*, have a direct impact on the computational efficiency of the overall method: The decomposition influences the *number* of solver calls, and the complexity of the subproblems has a decisive effect on the computational time required by *each individual* solver call. Different objective space methods have been proposed, e.g., in Klein and Hannan (1982); Sylva and Crema (2004); Laumanns et al. (2006); Sylva and Crema (2008); Lokman and Köksalan (2013); Özlen et al. (2014); Kirlik and Sayin (2014); Dächert and Klamroth (2015); Klamroth et al. (2015); Boland et al. (2016); Dächert et al. (2017); Boland et al. (2017b); Holzmann and Smith (2018); Tamby (2018); Turgut et al. (2019); Joswig and Loho (2020) and Tamby and Vanderpooten (2020). We review the most prominent approaches in the light of a generic algorithmic description in Sect. 4.

In the following subsections we first discuss the intrinsic difficulty of MOIP problems (Sect. 1.1) and then give a formal problem formulation and introduce the notation (Sect. 1.2). We provide a prototypical formulation of a *generic scalarization-based algorithm* in Sect. 2, discuss the geometric complexity of the associated decomposition, and briefly review scalarization methods that can be used in this context. Section 3 presents how the number of integer problems can be reduced when using the ε-constraint scalarization. In Sect. 4 we review the related literature and provide a summary of the leading objective space algorithms for MOIP. An extensive computational study comparing different approaches on instances of knapsack, assignment travelling salesman problems with up to ten objective functions is presented in Sect. 5. While the approach presented in this paper does not necessarily require the fewest number of integer programs to be solved, it is, nevertheless, the best with respect to CPU time.

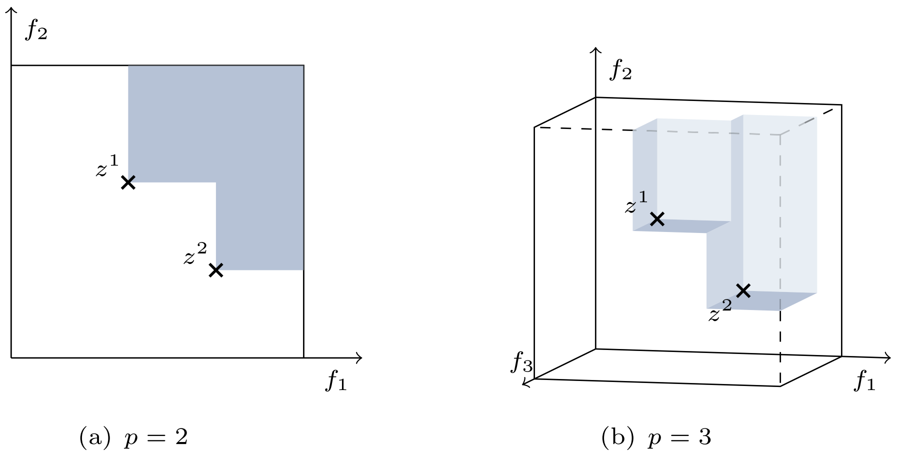

Fig. 1 Dominated region in the biobjective (**a**) and in the three-objective case (**b**) for two points $z^1 = (4, 6)$ and $z^2 = (7, 3)$ (**a**) and $z^1 = (4, 6, 6)$ and $z^2 = (7, 3, 4)$ (**b**), respectively. Note that **a** can be interpreted as a projection of **b** onto the $f_1 - f_2$-plane

1.1 Challenges in multiobjective integer programming

While MOIP problems are relatively well understood in the biobjective case, they are principally difficult in three and more dimensions (see, for example, Figueira et al. 2017). One reason for the specific role of biobjective problems is that in this case Pareto optimal solutions can be sorted such that their objective values are increasing in one objective and decreasing in the other objective. In this way, a natural ordering within the nondominated set is induced which facilitates central operations like, for example, decomposition, bound computations, and filtering for dominated solutions. This is used in the context of objective space methods, for example, in Aneja and Nair (1979); Chalmet et al. (1986); Ralphs et al. (2006). Since there is not such a natural ordering in the case of three and more objectives, many approaches do not directly transfer from the biobjective to the multiobjective case. Figure 1 illustrates this situation with the help of a simple example with two nondominated outcome vectors in the biobjective case (Fig. 1a) and in the three-objective case (Fig. 1b).

In addition, when increasing the number of objective functions then also the number of efficient solutions usually increases. Even though already biobjective problems are *intractable* in the sense that the size of the nondominated set may grow exponentially with the problem size (see, for example, Ehrgott 2005), in practice the percentage of nondominated outcome vectors largely depends on two model characteristics: The problem structure and coefficients, and the number of objective functions. Indeed, when extending an MOIP problem by an additional objective function, then all formerly efficient solutions remain efficient for the extended problem, and chances are high that further solutions become efficient if they perform well w.r.t. the additional objective. It is easy to see that when the solution set is finite it is always possible to define a finite number of objective functions so that *all* feasible solutions are efficient.

1.2 Problem formulation and notation

In the following, we give a brief introduction to multiobjective optimization and to the classical concept of Pareto dominance. For an extensive introduction into the field, we refer to the textbooks (Ehrgott 2005) and Miettinen (1999). Throughout this paper we use the following notation to compare two vectors $x^1, x^2 \in \mathbb{R}^n$:

$$\begin{aligned} x^1 &\leqq x^2 \Leftrightarrow x_i^1 \leq x_i^2 \text{ for all } i = 1, \ldots, n \\ x^1 &\leq x^2 \Leftrightarrow x_i^1 \leq x_i^2 \text{ for all } i = 1, \ldots, n \text{ and } x^1 \neq x^2 \\ x^1 &< x^2 \Leftrightarrow x_i^1 < x_i^2 \text{ for all } i = 1, \ldots, n. \end{aligned}$$

The symbols $\geqq$, $\geq$ and $>$ are used analogously.

We consider *multiobjective integer programming problems* (MOIP) with $p \geq 2$ functions given by

$$\min\{Cx \,:\, Ax \leqq b,\, x \in \mathbb{Z}^n\}, \tag{MOIP}$$

where $C \in \mathbb{Z}^{p \times n}$ is the objective matrix, $A \in \mathbb{Z}^{m \times n}$ is the constraint matrix, $b \in \mathbb{Z}^m$ is the right-hand-side vector, and $X = \{x \in \mathbb{Z}^n \,:\, Ax \leqq b\}$ denotes the set of feasible solutions of MOIP. We assume that all coefficients are integer and that the variables may only take integer values. Non-negativity constraints can be included in the constraint system $Ax \leqq b$ when required. We note that *multiobjective binary programming problems* (MOBP) where the solution vectors are constrained to $x \in \{0, 1\}^n$ are an important special case that is included in this formulation.

The p objective functions are given as a p-dimensional vector $f = (f_1, \ldots, f_p) : X \to \mathbb{Z}^p$, where $p \geq 2$ denotes the number of objective functions and hence the dimension of the objective space. The ith objective function is given as a linear function with cost vector $c^i = (c_{i1}, \ldots, c_{in})$, i.e., $f_i(x) = c^i x = \sum_{j=1}^n c_{ij} x_j$, $i = 1, \ldots, p$. Since we assume that all coefficients are integer, the set of attainable (i.e., feasible) outcome vectors $Z = f(X)$ satisfies $Z \subseteq \mathbb{Z}^p$.

We focus on the determination of *Pareto optimal* (or *efficient*) solutions which are those solutions for which none of the objective values can be improved without deterioration of at least one other objective value. This concept is based on the partial ordering induced by the natural ordering cone $\mathbb{R}^p_{\geq} = \{z \in \mathbb{R}^p \,:\, z \geq 0\} = \mathbb{R}^p_{\geqq} \setminus \{0\}$, where $0 \in \mathbb{R}^p$ denotes the p-dimensional zero-vector. Then $z^1 \leq z^2$ (i.e., z^1 *dominates* z^2) if and only if $z^1 \in z^2 - \mathbb{R}^p_{\geq}$, and $\bar{x} \in X$ is Pareto optimal if and only if there does not exist $x \in X$ such that $f(x) \in f(\bar{x}) - \mathbb{R}^p_{\geq}$.

The image of a Pareto optimal solution in the objective space is called *nondominated point* or *nondominated outcome vector*. The set of all Pareto optimal solutions (nondominated points) is referred to as the *Pareto set* (*nondominated set*) and denoted by $X_E \subseteq X$ and $Z_N \subseteq Z$, respectively. For a given nonempty set $N \subseteq Z$ of feasible outcome vectors, we refer to the set $N + \mathbb{R}^p_{\geqq}$ as the *dominated region* of the set N, see Fig. 1 for an illustration. Note that further nondominated outcome vectors of MOIP can only be located in the complement of the dominated region of the set N, i.e., in the set $S(N) := \mathbb{R}^p \setminus (N + \mathbb{R}^p_{\geqq})$ which we call the *search region w.r.t. N*. This is in fact true for every nonempty set $N \subseteq Z$, irrespective of the fact whether all points in N are nondominated or not. In the special case that $N = Z_N$ we additionally have that $N + \mathbb{R}^p_{\geqq} = Z + \mathbb{R}^p_{\geqq}$ and that $S(N) \cap Z = \emptyset$.

2 Generic scalarization-based algorithm

In this section, we provide a generic formulation of an objective space algorithm that is based on an appropriate decomposition of the search region and on the iterative solution of scalarized single-objective IPs using standard IP solvers. We emphasize that this method does not assume a specific (combinatorial) problem structure and that it takes advantage of the efficiency of readily available IP solvers.

Given an initial area of interest, i.e., an initial search region, each solver call either generates a new nondominated outcome vector (which, by removing the region it dominates, leads to a reduction of the search region) or returns the information that the corresponding subproblem is infeasible (and that the associated part of the search region does not contain any further nondominated outcome vectors and can be excluded from further evaluations).

Key operations are hence the description and update operation for the search region as the basis for its decomposition into *search zones* (Sect. 2.1) and the formulation of appropriate subproblems in the respective search zones which is usually realized through scalarizations (Sect. 2.2). A pseudocode formulation of the general algorithmic framework is given in Algorithm 1.

Algorithm 1: Generic Scalarization-Based Algorithm

```
Input  : MOIP problem
Output: Set of nondominated points $Z_N$
1  Determine initial search region
2  while exists unexplored search zones of the search region do
3      Choose search zone, solve subproblem 'therein'
4      if subproblem infeasible then
5          remove explored search zone from search region
6      else
7          save new nondominated point $z^s$
8          update search region based on $z^s$
9      end
10 end
```

As said before, the driving factors for the efficiency of Algorithm 1 are the *number* and the *complexity* of subproblems that are solved during the course of the algorithm. These two factors are interrelated: By formulating more complex subproblems, larger reductions of the search region can be achieved which can in turn be used to reduce the number of solver calls. However, this usually comes at the price of more expensive solver calls. This trade-off is analyzed in the numerical tests in Sect. 5. We note that an additional criterion when implementing Algorithm 1 is the avoidance of infeasible subproblems since detecting infeasibility often comes at a (slightly) higher computational cost.

2.1 The search region and its complexity

In this section we first focus on the mathematical description of the search region and show how it suggests a *natural decomposition* into non-redundant search zones. Towards this end, consider an arbitrary iteration of Algorithm 1 and let $N \neq \emptyset$ denote the set of nondominated points computed so far. Then the search region $S(N)$ that potentially contains *further* nondominated points is formally defined as

$$\begin{aligned} S(N) &= \mathbb{R}^p \setminus (N + \mathbb{R}^p_{\geqq}) \\ &= \mathbb{R}^p \setminus \{z \in \mathbb{R}^p : \exists \bar{z} \in N \text{ with } \bar{z} \leqq z\}. \end{aligned}$$

A thorough analysis of the geometric structure of the search region and its concise description is given in Dächert and Klamroth (2015) for the three-objective case, and in Klamroth et al. (2015) and Dächert et al. (2017) for the general case. See also Joswig and Loho (2020) for an alternative yet equivalent interpretation based on monomial cones and tropical algebra.

We follow the exposition in Klamroth et al. (2015) and assume that the search region is restricted to a bounding box with lower bound $(m, \ldots, m)^T \in \mathbb{Z}^p$ and upper bound $(M, \ldots, M)^T \in \mathbb{Z}^p$ (with $m < M$). These bounds may be given by the decision maker or they may be derived, for example, from global bounds on the attainable objective function values. For example, the *ideal point* can be used to determine a global lower bound. For an MOIP problem the ideal point $z^I \in \mathbb{Z}^p$ is defined component-wise as

$$z^I_i = \min\{c^i x : Ax \leqq b, \ x \in \mathbb{Z}^n\}, \ i = 1, \ldots, p. \tag{1}$$

To simplify the notation, we assume wlog that $m = 0$. Moreover, let p *dummy points* $d^i = M \cdot e^i \in \mathbb{Z}^p, i = 1, \ldots, p$ be included in the set N that delimit the bounding box, where e^i denotes the ith unit vector in $\mathbb{R}^p$. Then the search region $S(N)$ can be unambiguously described by a finite set of *local upper bounds* $U(N)$ such that

$$\begin{aligned} S(N) &= \{z \in \mathbb{R}^p : \exists u \in U(N), \ z < u\} \\ &= \textstyle\bigcup_{u \in U(N)} C(u), \end{aligned}$$

where $C(u) = u - \mathbb{R}^p_{>}$ denotes the *search zone* induced by the local upper bound u. Figure 2 illustrates the situation for the example problems from Fig. 1.

Intuitively, local upper bounds are the maximal points in the objective space that are not dominated by any of the points in N. As a consequence, all components $u_i, i = 1, \ldots, p$, of a local upper bound $u \in U(N)$ are induced by a corresponding component z^i_i of a point $z^i \in N$ that satisfies $z^i_i = u_i$ and $z^i_j < u_j$ for all $j \in \{1, \ldots, p\} \setminus \{i\}$. We call such a point a *defining point* for (component i of) the local upper bound u. Note that local upper bounds are thus also integer points, i.e., $U(N) \subset \mathbb{Z}^p$.

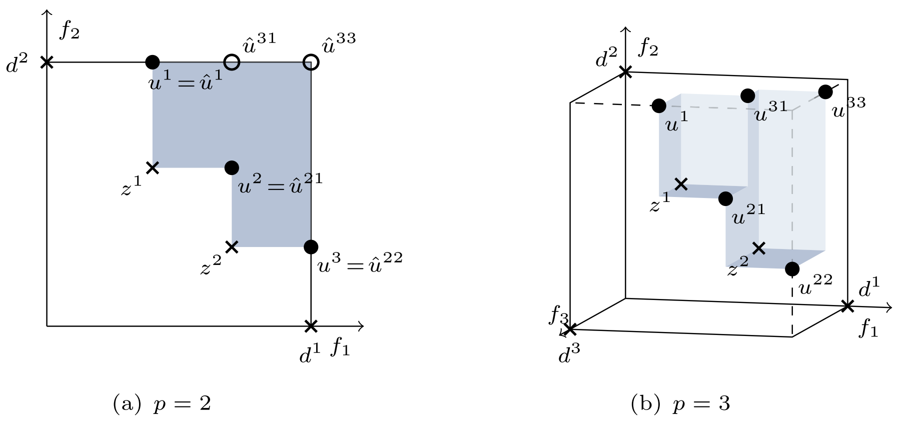

Fig. 2 Local upper bounds shown by filled circles in the biobjective (**a**) and in the three-objective case (b) for the two points introduced in Fig. 1. The empty circles in **a** show the projections $\hat{u}^{31}$ and $\hat{u}^{33}$ of the local upper bounds u^{31} and u^{33} that are only present in the three-objective case, see **b**

Moreover, as soon as at least one nondominated point z in the interior of the bounding box is found (this is usually the case after solving the first subproblem), then the global upper bound $(M, \dots, M)^T$ is dominated and every local upper bound has at least one defining point that is not a dummy point.

More generally, whenever a new nondominated point $\bar{z}$ is found in $U(N)$, then the set $U(N)$ needs to be updated: For all $u \in U(N)$ with $\bar{z} < u$, u is replaced by p smaller local upper bounds $u^1, \dots, u^p$, where

$$u^i_j = \begin{cases} \bar{z}_j, & j = i \\ u_j, & j \neq i \end{cases} \qquad \text{for } i = 1, \dots, p. \tag{2}$$

I.e., to compute u^i, the i-th component of u is replaced by the value of the nondominated point $\bar{z}$ in component i while all other components remain unchanged. Hence, $u^i \leq u$ for all $i = 1, \dots, p$, that is, the search region is reduced. For convenience we call the resulting local upper bound u^i an *i-child of u*. Note that $\bar{z} < u$ may be satisfied for more than one local upper bound in $U(N)$. In this case, some of the generated i-children may be redundant for the description of $U(N \cup \{\bar{z}\})$ and can be removed. The detection of redundant local upper bounds is described in detail in Dächert and Klamroth (2015), Klamroth et al. (2015) and Dächert et al. (2017).

In the biobjective case (see Fig. 2a for an illustration), the set $U(N)$ is immediately obtained from the points in N by first sorting these points, for example, w.r.t. f_1, and then combining their respective components in increasing order of f_1 (in Fig. 2a, this yields $U(N) = \{u^1, u^2, u^3\}$). Figure 2b illustrates the set $U(N) = \{u^1, u^{21}, u^{22}, u^{31}, u^{33}\}$ for an example problem with three objectives that was computed by recursively introducing the outcome vectors z^1 and z^2. Efficient operations to iteratively update the set $U(N)$ when further nondominated points are added to N are described in Klamroth et al. (2015) and Dächert et al. (2017).

Given the set $U(N)$, a single-objective subproblem can be solved for each of the associated *search zones* $C(u) = u - \mathbb{R}^p_>$ to either find a new (mutually nondominated) point, or to show that the corresponding search zone does not contain any further nondominated points. The cardinality of the set $U(N)$ is thus decisive for the *number* of subproblems that need to be solved in Algorithm 1. A worst case bound on $|U(N)|$ can be derived from results of Boissonnat et al. (1998) and Kaplan et al. (2008) from the field of algorithmic geometry: The number of local upper bounds is bounded by

$$|U(N)| = O\left(|N|^{\lfloor \frac{p}{2} \rfloor}\right) \qquad \text{for } p \geq 2. \tag{3}$$

This bound is tight in the sense that there are instances where this bound is attained. However, in practice the value of $|U(N)|$ is often much smaller, often even by orders of magnitude. In the special case of two objectives ($p = 2$) the number of local upper bounds is exactly $|U(N)| = |N| + 1$, irrespective of the considered problem instance. Moreover, Dächert and Klamroth (2015) showed that for three objectives ($p = 3$) it holds that $|U(N)| = 2|N| + 1$ (under the assumption that no two points in N share the same value in one of their components). Hence, for $p = 3$ the number of search zones still grows only linearly with the number of nondominated points.

We argue that the decomposition of the search region $S(N)$ into $|U(N)|$ pairwise non-redundant search zones $C(u)$, $u \in U(N)$, is a natural decomposition that immediately leads to the formulation of associated pairwise non-redundant single-objective subproblems. The efficiency of this method is validated in the numerical tests presented in Sect. 5. While other decompositions of the search region are of course possible and have other pros and cons, the structure of the set $U(N)$ and particularly its cardinality remain decisive for the computational complexity of the respective methods.

Indeed, when the number of objective functions p is fixed and when the natural decomposition is combined with a suitable scalarization method, then (3) implies a polynomial bound on the overall number of iterations of Algorithm 1. In this situation we call a scalarization method *suitable* if and only if either a new nondominated point is found or the respective search zone is identified as an empty zone (which can be excluded from further consideration) when solving an appropriate scalarization in a search zone. This property is, e.g., satisfied by the frequently applied ε-constraint scalarization and by the weighted Tchebychev scalarization, see Sect. 2.2 below for further details.

Theorem 1 *When the natural decomposition of the search region into non-redundant search zones is combined with a suitable scalarization method, then the overall number of iterations of Algorithm 1 can be bounded by*

$$O\left(|Z_N|^{\lfloor \frac{p}{2} \rfloor}\right) \qquad \textit{for } p \geq 2, \tag{4}$$

which is polynomial in the size of the nondominated set $|Z_N|$ when p is fixed.

Proof First consider the case where the $|Z_N|$ nondominated points are found within the first $|Z_N|$ iterations, and after that the remaining search zones are investigated which are empty. In this case, the number of subproblems solved by Algorithm 1 is bounded by

$$O\left(|Z_N| + |Z_N|^{\lfloor \frac{p}{2} \rfloor}\right) = O\left(|Z_N|^{\lfloor \frac{p}{2} \rfloor}\right) \qquad \text{for } p \geq 2, \tag{5}$$

i.e., $|Z_N|$ iterations in each of which a nondominated point is determined, and at most $O(|Z_N|^{\lfloor \frac{p}{2} \rfloor})$ iterations to detect that all search zones induced by these points are empty. C.f. (3) for the corresponding bound on the number of search zones induced by Z_N.

Now consider the general case where empty search zones may be detected at any time. In this case the above argumentation still holds since an empty search zone is never changed in later iterations, irrespective from the fact whether it is detected earlier or later during the procedure. Indeed, only those search zones are updated and split that contain a newly detected nondominated point, and this situation can not occur when the considered search zone is empty. Therefore, the iteration at which an empty search zone is detected does not matter. □

Note that Theorem 1 relies on two central properties: (a) The complexity of the decomposition used in Algorithm 1 grows only polynomially with the number of nondominated points (which is guaranteed in the case of the natural decomposition), and (b) the employed scalarization must be tailored for the respective search zones in the sense that with each solver call, either a new nondominated point is found or the search zone is excluded. We will focus on this property when discussing suitable scalarizations, tailored for the natural decomposition, in the following section.

To the best of our knowledge the above approach is the only objective space method that has a provably polynomial bound on the number of required solver calls while keeping these solver calls simple, i.e., without relying on disjunctive constraints and/or additional integer variables.

2.2 Scalarization methods

We now assume that an intermediate set N of already computed nondominated points and a decomposition of the associated search region into a finite set of search zones is given. We exemplify the following discussion at the natural decomposition described in Sect. 2.1 above, where each search zone $C(u)$, $u \in U(N)$, corresponds to a hyper-rectangle with lower bound 0 and upper bound u.

Algorithm 1 then iteratively selects a search zone in order to decide whether it is feasible and hence contains further nondominated points–in this case, such a nondominated point has to be computed–or whether it is infeasible and can be excluded from further consideration. This is realized by formulating an appropriate single-objective subproblem. For search zones with a simple structure as in the case of the natural decomposition, this can be realized by appropriate *scalarizations*, see, for example, Ehrgott (2006) for a survey in the context of multiobjective integer programming problems. In the following, we focus on ε-constraint scalarizations and weighted Tchebychev scalarizations since they are particularly well suited in this setting.

2.2.1 ε-constraint scalarization

The most commonly used scalarization in this context is the ε-*constraint scalarization*: Given a search zone $C(u)$ with local upper bound u, the associated ε-constraint subproblem w.r.t. the ith objective, $i \in \{1, \dots, p\}$, is given by

$$\min_{x \in X} \left\{ c^i x \ : \ c^j x \leq u_j - 1 \text{ for } j \in J_i \right\}, \qquad (P_i^\varepsilon(u))$$

where $J_i := \{1, \dots, p\} \setminus \{i\}$ denotes the index set of all non-selected objective functions. Figure 4a illustrates the additional constraints together with an exemplary level curve of the objective function. The following lemma illuminates the role of the defining points of the local upper bound u for the associated ε-constraint scalarization ($P_i^\varepsilon(u)$). We refer to Tamby and Vanderpooten (2020) for a similar result.

Lemma 1 *Let $C(u)$ be a search zone from the natural decomposition of the search region, let $i \in \{1, \dots, p\}$ be a selected objective function such that u_i is not defined by a dummy point, and let z^i be a defining point for the i-th component of u with pre-image $x^i \in X$. Then it holds:*

1. *x^i is a feasible solution for the ε-constraint scalarization ($P_i^\varepsilon(u)$).*
2. *If x^i is an optimal solution of ($P_i^\varepsilon(u)$), then the search zone $C(u)$ does not contain any further nondominated points.*

Proof Since $z^i \in N \setminus \{d^1, \dots, d^p\}$ is a defining point of u we have that $z^i_j < u_j$ for all $j \in J_i$. Since in addition $z^i = Cx^i$ with $x^i \in X$, it follows that $z^i \in \mathbb{Z}^p$ and hence $z^i_j = c^j x^i \leq u_j - 1$ for all $j \in J_i$. Thus, x^i is feasible for problem ($P_i^\varepsilon(u)$).

Now let x^i be an optimal solution of problem ($P_i^\varepsilon(u)$). Then there is no $\bar{x} \in X$ with $c^j \bar{x} < u_j$, $j \in J_i$ (recall that $c^j \bar{x}$ is integral for all $j \in \{1, \dots, p\}$) and $c^i \bar{x} < z^i_i = u_i$, and hence $Z \cap C(u) = \emptyset$. $\square$

If the ε-constraint scalarization ($P_i^\varepsilon(u)$) is solved by an IP-solver that can be sped up when providing a feasible starting solution, then Lemma 1 suggests to select a minimization objective f_i, $i \in \{1, \dots, p\}$, such that the corresponding defining point z^i is not a dummy point since then a corresponding pre-image $x^i \in X$ with $Cx^i = z^i$ is already known and can be used as a feasible starting solution when solving problem ($P_i^\varepsilon(u)$). Moreover, problem ($P_i^\varepsilon(u)$) is then feasible which usually has a positive effect on the computational time (as compared to infeasible subproblems).

The following properties are well-known; we refer to the textbook (Ehrgott 2005) for further details.

Lemma 2 *Every optimal solution $\bar{x} \in X$ of an ε-constraint scalarization ($P_i^\varepsilon(u)$) is weakly efficient for MOIP, and at least one of the optimal solutions of ($P_i^\varepsilon(u)$) is efficient for MOIP.*

When combining the results of Lemmas 1 and 2 we can conclude that if an ε-constraint scalarization ($P_i^\varepsilon(u)$) returns an optimal solution $\bar{x} \in X$ with $c^i \bar{x} < z^i_i$, then this solution is in the search zone $C(u)$ and it is at least weakly efficient for ($P_i^\varepsilon(u)$). In this case, $Z_N \cap C(u) \neq \emptyset$ and a nondominated point in $C(u)$ can be found by, for example, performing a lexicographic optimization over the set of optimal solutions of problem ($P_i^\varepsilon(u)$). The latter can be realized, e.g., by solving a *second-stage problem*

$$\min_{x \in X} \left\{ \sum_{j=1}^{p} c^j x \ : \ c^i x = c^i \bar{x},\ c^j x \leq u_j - 1 \text{ for } j \in J_i \right\} \qquad (P_i^{\varepsilon,2}(u))$$

over the set of optimal solutions of problem ($P_i^\varepsilon(u)$). Note that a feasible starting solution is readily available by the optimal solution from the first stage problem.

Note that this *two-stage approach* requires two solver calls whenever a search zone contains further nondominated points. This can be avoided by adding an appropriate augmentation term to problem ($P_i^\varepsilon(u)$), leading to an *augmented ε-constraint scalarization*

$$\min_{x \in X} \left\{ c^i x + \rho \sum_{j=1}^{p} c^j x \ : \ c^j x \leq u_j - 1 \text{ for } j \in J_i \right\}, \qquad (P_i^{\varepsilon,\rho}(u))$$

which returns an efficient solution of MOIP. Here, $\rho > 0$ is an appropriately chosen small *augmentation parameter*. We refer to Ehrgott and Ruzika (2008) for a detailed discussion on augmented ε-constraint scalarizations and appropriate choices of ρ.

Remark 1 Since an optimal solution $\bar{x}$ of the ε-constraint scalarization ($P_i^\varepsilon(u)$) minimizes the selected objective function f_i in the search zone $C(u)$, there can not be any further nondominated points in $C(u)$ that have an even better value in objective f_i. As a consequence, when updating the decomposition w.r.t. the new point $\bar{z} = C\bar{x}$, search zones $C(\bar{u})$ with $\bar{u} \leq u$ and $\bar{u}_i \leq \bar{z}_i$ are known to be empty and can be removed from the search region without further consideration. Depending on the applied decomposition, this fact usually results in a reduction of solver calls and is thus an advantage of using ε-constraint scalarizations. In the example depicted in Fig. 3, $u^{11} \leq u^1$ and $u_1^{11} \leq z_1^3$, hence, u^{11} can be removed without further inspection. We discuss the realization of such a reduction in detail in Sect. 3 below.

2.2.2 Weighted Tchebychev scalarization

The weighted Tchebychev scalarization minimizes a weighted Tchebychev distance $l_\infty^w(z, z^r) = \max_{j=1,\dots,p} |w_j(z_j - z_j^r)|$ between a reference point z^r and the closest feasible point $z \in Z$ in the objective space.

In order to determine whether a given search zone $C(u)$ with $u \in U(N)$ contains further nondominated points, a natural choice for the reference point is the origin, i.e., $z^r = 0$. The weights $w_j > 0$, $j = 1, \dots, p$ of the weighted Tchebychev distance are

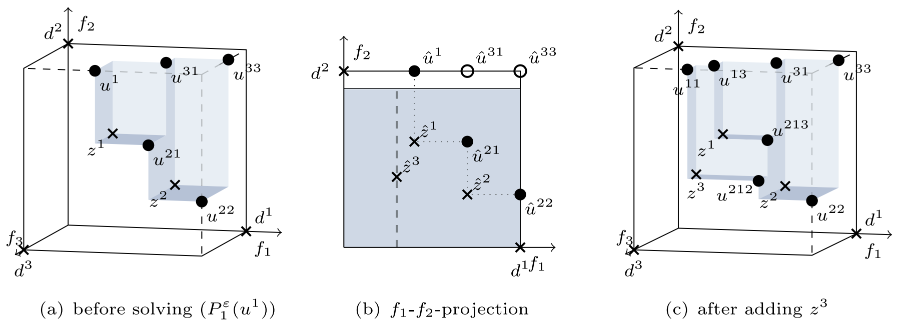

Fig. 3 **a** The 3d situation from Fig. 2b before solving the next subproblem; **b** 2-dimensional projection of the ε-constraint scalarization $(P_1^{\varepsilon}(u^1))$ in $C(u^1)$ minimizing f_1 with feasible set constrained by $c^j x \leq u_j^1 - 1$, $j = 2, 3$ (shaded) and level curve associated to an optimal point $z^3 = (3, 4, 8)$ (dashed) for the problem shown in **a**; **c** 3d situation with updated local upper bounds for the same problem after adding the newly generated point $z^3 = (3, 4, 8)$ to the set N

then set such that the level set of level 1 passes through all defining points of u and has its upper right corner in u, see Fig. 4b for an illustration. This yields $w_j = \frac{1}{u_j}$, $j = 1, \dots, p$. Note that we assume wlog that $u > 0$ for all $u \in U(N)$ so that these weights are well-defined. Since we assume that the bounding box is contained in $\mathbb{R}^p_{\geqq}$ we can omit all absolute values and formulate the *weighted Tchebychev scalarization* w.r.t. the search zone $C(u)$ as

$$\min_{x \in X} \left\{ \max_{j=1,\dots,p} w_j\, c^j x \right\} = \min_{x \in X} \left\{ \alpha \;:\; c^j x \leq \alpha u_j \text{ for } j = 1, \dots, p,\ \alpha \in \mathbb{R} \right\}. \tag{$P^T(u)$}$$

Note that the second, constrained reformulation can be immediately passed to a standard IP-solver.

Lemma 3 *Let $C(u)$ be a search zone from the natural decomposition of the search region. Then it holds:*

1. *Every pre-image $x^i \in X$ of a defining point z^i of u, $i \in \{1, \dots, p\}$, that is not a dummy point is a feasible solution for the weighted Tchebychev scalarization $(P^T(u))$ with objective value 1.*
2. *If the optimal objective value of problem $(P^T(u))$ is 1, then the search zone $C(u)$ does not contain any further nondominated points.*

Proof Since every solution in X is feasible for problem $(P^T(u))$ this is also true for every pre-image x^i of a defining point z^i of u, $i \in \{1, \dots, p\}$. Since $z_i^i = c^i x^i = u_i$ and $z_j^i = c^j x^i < u_j$ for all $j \in J_i$ it follows that $\max_{j=1,\dots,p} w_j\, c^j x^i = \max_{j=1,\dots,p} \frac{c^j x^i}{u_j} = \frac{c^i x^i}{u_i} = 1$.

Now suppose that the optimal objective value of problem $(P^T(u))$ is 1. Then there is no solution $\bar{x} \in X$ that satisfies $c^j x < u_j$ for all $j = 1, \dots, p$, and hence $Z \cap C(u) = \emptyset$. □

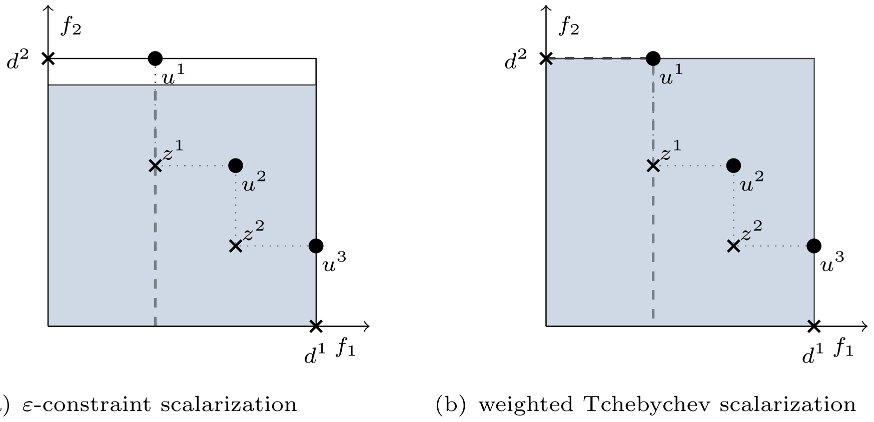

Fig. 4 (a) ε-constraint scalarization in $C(u^1)$ minimizing f_1 with feasible set constrained by $f_2(x) = c^2x \leq u^1_2 - 1$ (shaded) and level curve (dashed) through z^1, which can be used as feasible starting solution; (b) weighted Tchebychev scalarization in $C(u^1)$ with feasible set (shaded) and level curve (dashed) passing through u^1, using also z^1 as feasible starting solution

Note that different from the ε-constraint scalarization ($P^\varepsilon_i(u)$), *all* previously determined solutions, and hence in particular all defining points of the considered search zone or its upper bound, respectively, that are not dummy points, can be used as (relatively good) feasible starting solutions for the weighted Tchebychev scalarization ($P^T(u)$).

In general, the theoretical properties of the weighted Tchebychev scalarization and of the ε-constraint scalarization are very similar. We refer again to the textbook (Ehrgott 2005) for the following result.

Lemma 4 *Every optimal solution $\bar{x} \in X$ of a weighted Tchebychev scalarization ($P^T(u)$) is weakly efficient for (MOIP), and at least one of the optimal solutions of ($P^T(u)$) is efficient for (MOIP).*

Similar to the case of the ε-constraint scalarization, weakly efficient solutions can be avoided by solving a *second-stage problem*

$$\min_{x \in X} \left\{ \sum_{j=1}^{p} c^j x \ : \ c^j x \leq \bar{\alpha}\, u_j \text{ for } j \in \{1, \dots, p\} \right\} \qquad (P^{T,2}(u))$$

over the set of optimal solutions of problem ($P^T(u)$), where $\bar{\alpha}$ denotes the optimal objective value of problem ($P^T(u)$). Again, a feasible starting solution for this second stage problem is known from solving the first stage problem.

Rather than solving two IP-problems, the original problem can also be extended by an augmentation term to the objective, yielding an *augmented weighted Tchebychev* *scalarization*

$$\min_{x \in X} \left\{ \alpha + \rho \sum_{j=1}^{p} c^j x \; : \; c^j x \leq \alpha u_j \text{ for } j = 1, \ldots, p, \; \alpha \in \mathbb{R} \right\}. \qquad (P^{T,\rho}(u))$$

We emphasize that the augmentation parameter $\rho > 0$ has to be chosen carefully. Indeed, while ρ has to be chosen small enough such that no efficient solution is missed, an overly small choice of ρ may lead to numerical problems. A detailed analysis with a concrete suggestion for an "optimal" choice of the augmentation parameter when $p = 2$ is given in Dächert et al. (2012).

Figure 4 illustrates the additional constraints added to the problem in the respective scalarizations (ε-constraint versus weighted Tchebychev). In addition, exemplary level curves of the respective objective functions are shown.

3 Reduction of the search region when using the ε-constraint method

If we use the ε-constraint scalarization in Line 3 of Algorithm 1, we can reduce the search region further. This is due to a specific property of the the ε-constraint scalarization which allows to remove one newly created search zone directly in every iteration in which a new nondominated point is found. However, to make use of this additional reduction, we cannot select the zone in which we search for a new nondominated point arbitrarily but have to select it according to some criterion. Note that the idea itself is not new but has already been elaborated in Dächert and Klamroth (2015) for the tricriteria case.

In the following we present a selection criterion that is valid for any number of criteria. We prove that this selection criterion allows to reduce the total number of evaluated search zones by the number of nondominated points. In the proof we make use of certain results from Dächert et al. (2017), namely Lemma 3.8 and Proposition 4.3, which we recall below before stating and proving the selection criterion. They use both the notion of a *neighbor* from Definition 3.6 in Dächert et al. (2017) which we present in a shorter form below.

Definition 1 [Neighbor] Two local upper bounds $u, u' \in U(N)$ are neighbors if they share $p-1$ defining points, among which exactly one changes its position from some $j \in \{1, \ldots, p\}$ to some $k \in \{1, \ldots, p\}$, $k \neq j$. That is, there are two indices j, k with $j \neq k$ such that $z^j(u) = z^k(u')$ while $z^i(u) = z^i(u')$ for all $i \in \{1, ..., p\} \setminus \{j, k\}$. We then say that u is a j-neighbor of u', and that u' is a k-neighbor of u.

Lemma 5 (Dächert et al. 2017) *Let $u, u' \in U(N)$ be local upper bounds such that u' is a k-neighbor of u, and u is a j-neighbor of u'. Then, we have $u_j < u'_j$, $u_k > u'_k$ and $u_i = u'_i$ for $i \neq j, k$.*

Proposition 1 (Dächert et al. 2017) *Let $u \in U_{\bar{z}} = \{u' \in U(N) : \bar{z} < u'\}$. Then, for every $k = 1, \ldots, p$, one of the following two assertions holds exclusively:*

1. $u^k \in U(N \cup \{\bar{z}\})$, where u^k denotes the k-child of u

2. $u' \in U_{\bar{z}}$, where u' is the k-neighbor of u.

Theorem 2 (Min-component selection criterion) *If, in every iteration of Algorithm 1, we select a search zone $\bar{u}$ which is minimal in some component $i \in \{1, \dots, p\}$, i.e., for which*

$$\bar{u}_i = \min\{u_i : u \in U(N)\} \tag{6}$$

for some $i \in \{1, \dots, p\}$ holds, and if we then solve an ε-constraint problem of the form $P_i^{\varepsilon,2}(\bar{u})$ or $P_i^{\varepsilon,\rho}(\bar{u})$, then, whenever there is a solution $\bar{z} \in Z_N$ with $\bar{z} < \bar{u}$, the i-child of $\bar{u}$ exists, i.e., $\bar{u}^i \in U(N \cup \{\bar{z}\})$. Moreover, $\bar{u}^i$ can not contain further nondominated points, i.e., $\{z \in Z_N : z < \bar{u}^i\} = \emptyset$.

Proof We first show that $\bar{u}^i \in U(N \cup \{\bar{z}\})$. Since $\bar{z} < \bar{u}$, using the notation of Proposition 1, $\bar{u} \in U_{\bar{z}}$ holds.

Assume that $\bar{u}^i \notin U(N \cup \{\bar{z}\})$. Then, due to Proposition 1, the i-neighbor u' of $\bar{u}$ contains $\bar{z}$. According to Lemma 5, $u'_i < \bar{u}_i$ holds for the i-neighbor u' of $\bar{u}$. However, this is a contradiction to $\bar{u}$ being minimal with respect to component i. Hence, $\bar{u}^i \in U(N \cup \{\bar{z}\})$ must hold.

It remains to show that $\bar{u}^i$ does not contain further nondominated points. Assume that there is a $z' \in Z_N$ with $z' < \bar{u}^i$. Then, in particular, $z'_i < \bar{u}^i_i = \bar{z}_i$ would hold. This, however, is a contradiction to $\bar{z}$ being optimal for the ε-constraint problem $P_i^{\varepsilon,2}(\bar{u})$ or $P_i^{\varepsilon,\rho}(\bar{u})$. □

Thanks to Theorem 2 we can remove one search zone per iteration, in which a new nondominated point is detected, when using the ε-constraint scalarization. The presented selection criterion is simple but not the only possible one. Another one is presented as a heuristic in Tamby and Vanderpooten (2020) (see their Section 3.3). In the following we formally prove that also their criterion yields the desired result of avoiding to solve unnecessary scalarizations.

Tamby and Vanderpooten (2020) select a search zone u^* by computing

$$(i^*, u^*) = \begin{cases} (1, (M, \dots, M)) & \text{if } N = \emptyset, \\ \underset{\substack{u \in U(N), \\ i \in \{1,\dots,p\}, \\ u_i \neq M}}{\arg\max} \{h(i, u)\} & \text{otherwise,} \end{cases} \tag{7}$$

where

$$h(i, u) = \prod_{j \neq i} \left(u_j - z_j^I\right)$$

and z^I denotes the ideal point, see (1). The index i^* serves to make use of starting solutions. If we ignore this additional feature, their selection criterion for a fixed index

i reads

$$u^* = \begin{cases} (M, \dots, M) & \text{if } N = \emptyset, \\ \underset{\substack{u \in U(N), \\ u_i \neq M}}{\arg\max} \left\{ \prod_{j \neq i} \left(u_j - z^I_j \right) \right\} & \text{otherwise,} \end{cases} \tag{8}$$

for a certain $i \in \{1, \dots, p\}$. Note that, except in the first iteration, a product consisting of $p - 1$ terms has to be computed for every search zone with $u_i \neq M$ to find the search zone with the largest projected hypervolume.

Theorem 3 [Max-projected-volume selection criterion] *If, in every iteration of Algorithm 1, we select a search zone $\bar{u}$ according to (8), and if we then solve an ε-constraint problem of the form $P_i^{\varepsilon,2}(\bar{u})$ or $P_i^{\varepsilon,\rho}(\bar{u})$, then, whenever there is a solution $\bar{z} \in Z_N$ with $\bar{z} < \bar{u}$, the i-child of $\bar{u}$ exists, i.e., $\bar{u}^i \in U(N \cup \{\bar{z}\})$. Moreover, $\bar{u}^i$ can not contain further nondominated points, i.e., $\{z \in Z_N : z < \bar{u}^i\} = \emptyset$.*

Proof The proof is similar to the proof of Theorem 2. Assume that $\bar{u}^i \notin U(N \cup \{\bar{z}\})$. Then, due to Proposition 1, the i-neighbor u' of $\bar{u}$ contains $\bar{z}$. According to Lemma 5, we then have $u'_i < \bar{u}_i$, $u'_k > \bar{u}_k$ for some $k \neq i$ and $u'_j = \bar{u}_j$ for $j \neq i, k$. However, then

$$\prod_{j \neq i} \left(u'_j - z^I_j \right) > \prod_{j \neq i} \left(\bar{u}_j - z^I_j \right)$$

would hold, a contradiction to $\bar{u}$ satisfying (8). Hence, $\bar{u}^i \in U(N \cup \{\bar{z}\})$ must hold.

The rest of the proof is the same as for Theorem 2. □

Theorem 3 demonstrates that the heuristic selection criterion of Tamby and Vanderpooten (2020) has indeed the same beneficial property as our selection criterion (6). However, ours is much simpler, since it is solely based on one component value of a search zone, an information that is available anyway. It does not require any computation of additional figures as, e.g., the projected hypervolumes computed in Tamby and Vanderpooten (2020).

As done in (7), we could additionally choose the index i which then determines the index to be optimized in the ε-constraint problem. Then, previously generated nondominated points can be used as starting solutions as discussed in Lemma 1.

4 Search strategies in objective space methods

Objective space methods can be distinguished w.r.t. the way in which they search for nondominated outcome vectors in the objective space and, in particular, in the search region. While the previous sections focused on a formal description of the search region and its decomposition into (rectangular) search zones, which naturally implies decomposition-based methods, we will also review alternative search strategies and provide a brief summary of some of the most successful objective space methods.

We classify these methods into three categories: Decomposition, recursive dimension reduction and disjunctive constraints. However, some methods can not be assigned clearly to one of these categories, wherefore we also discuss hybrid approaches in the end.

4.1 Decomposition

The concept of decomposition relies on the idea that the search region can be described by the union of rectangular sets which in turn can be described based on already generated nondominated points. In every iteration, one of these rectangular sets can be investigated which results in a rather simple IP to be solved. A natural and efficient decomposition of the search region was described in Sect. 2.1 above. However, there are different ways of decomposing the search region which will be briefly reviewed in the following.

One way consists in subdividing the search region into disjoint cells each of which is defined by a lower and upper bound vector. When a new nondominated point is generated, its components are inserted as axis-parallel "hyper-cuts" to all (concerned) cells. Dominated cells can be deleted immediately. Nevertheless, the number of remaining cells grows quickly. The union of all disjoint cells represents the search region. This idea is used in Laumanns et al. (2006, 2005), and Kirlik and Sayin (2014). The performance of an algorithm based on such a disjoint decomposition highly depends on the order in which the cells are investigated. Kirlik and Sayin (2014) select the box with the largest volume, where the volume is defined by the upper bound of the cell and the ideal point. Thereby they can avoid the generation of dominated points. However, without further enhancement strategies, it might happen that a nondominated point is computed more than once.

Other algorithms use a non-disjoint decomposition which leads to a smaller number of boxes to be saved and investigated. In this case, the boxes are usually only defined by upper bound vectors. Przybylski et al. (2010) use the idea of upper bounds in the second phase of their algorithm. They remove redundant upper bounds by a filtering step in every iteration. Lokman and Köksalan (2013) also implicitly use a sort of a non-disjoint decomposition by varying the right-hand side vectors of the constraints on the objective vectors in a systematic way based on all already generated nondominated points. Dächert and Klamroth (2015) propose a non-disjoint decomposition for tricriteria optimization problems which results in a number of upper bound vectors that depends linearly on the number of nondominated points. Klamroth et al. (2015) present a constructive algorithm that generates the set of local upper bound vectors based on a stable set of points, i.e., mutually nondominated points. The main idea is to use this stable set of points as defining points and to formulate a criterion which guarantees that the number of local upper bounds is minimal. This method will be referred to as the *defining point algorithm* (DPA) in Sect. 5 below. Another non-disjoint decomposition is proposed in Dächert et al. (2017) that is based on the inherent neighborhood structure between nondominated points and local upper bounds. Different to most other recursive and decomposition approaches, Dächert and Klamroth (2015), Klamroth et al. (2015) and Dächert et al. (2017) can guarantee that each nondominated point is only generated once during the algorithm which implies that no strategy based on the examination of all previously generated nondominated points is needed. Moreover, the DPA approach is not restricted to the application of ε-constraint scalarizations for the solution of subproblems. A specific implementation of the DPA approach is presented in Tamby and Vanderpooten (2020). It uses the particular properties of ε-constraint scalarizations in combination with a strategy for choosing the next search zone, which allows to use feasible starting solutions for the next IP to be solved. Holzmann and Smith (2018) suggest to use modified augmented weighted Tchebychev scalarizations to solve the subproblems, both in combination with simple non-disjoint decompositions including redundancies, and in combination with the DPA approach. Numerical tests confirm the clear superiority of the non-redundant decomposition from DPA.

4.2 Recursive dimension reduction

The idea of recursion is to reduce the dimension of the objective space (and hence the search region) until single criterion or bicriteria (sub-)problems are obtained. The results of lower level formulations are then used as bounds in the next higher level problem in a systematic way.

There are different ways how to organize the recursion. One way consists in selecting two objectives (typically the first and the second) and computing all points that are nondominated with respect to these two objectives. All other objectives are bounded from above, where the upper bounds are narrowed iteratively in every level of recursion. This is the basic idea of Chalmet et al. (1986), Özlen and Azizoğlu (2009) and Özlen et al. (2014).

Others also compute all possible combinations of recursions. Given a multiobjective optimization problem with p objectives they consider all p corresponding $(p-1)$-dimensional optimization problems and so on. Therefore, a tree is created with bicriteria optimization problems in its leaf nodes. It is shown in Ehrgott and Tenfelde-Podehl (2003) that the points obtained by solving all $(p-1)$-criteria problems represent a subset of the nondominated set of the original problem. This way of recursion has been used, e.g., in Tenfelde-Podehl (2003) and as a subprocedure in Dhaenens et al. (2010) and Przybylski et al. (2010).

A drawback of recursive algorithms is the fact that nondominated points are typically computed several times since they are often optimal for multiple recursive subproblems. Since the solution of IPs is costly, it is critical for the performance to avoid the repeated generation of already known nondominated points. In order to deal with this issue the information from all problems solved before can be stored. More precisely, this requires to save the right-hand side values of the bounded objectives as well as the result of the corresponding optimization run, i.e., the optimal solution of the problem or the information that it is infeasible. Before solving a new problem, the list of already solved problems is scanned. This idea is implemented in Özlen et al. (2014) and improved the number of solved IPs drastically.

4.3 Disjunctive constraints

Using disjunctive constraints is a general concept in optimization that is tailored for the case that the feasible set is non-convex. Typically, artificial binary variables are used to activate or deactivate parts of the feasible set. We refer, e.g., to the textbook Nemhauser and Wolsey (1999) for the general concept.

In the context of multiobjective optimization problems this concept is useful to describe the search region, which is a non-convex set. It is then possible to consider all remaining parts of the search region (that are not already known to be empty) simultaneously, i.e., to search for a new nondominated point by solving one single IP formulated over the complete (non-convex) search region. In this case, every nondominated point is computed exactly once. However, with every new nondominated point, the number of constraints and artificial binary variables increases which makes this approach computationally demanding and hardly ever competitive. Disjunctive constraints are used in Klein and Hannan (1982), Sylva and Crema (2004) and Lokman and Köksalan (2013).

4.4 Hybrid approaches

There are also hybrid methods that combine ideas from the three categories mentioned above. Boland et al. (2016) propose a method called L-shape search method for tricriteria problems that is a hybrid of disjunctive constraints and decomposition. The idea is to use disjunctive constraints only with respect to the lastly generated nondominated point which results in an L-shape element that is investigated with priority, i.e., shrunk quickly towards the ideal point. During the procedure, unexplored rectangular sets are saved and investigated in the later phase of the algorithm. Boland et al. (2017a) extend the L-shape search method to any number of objectives. The approach of Boland et al. (2017b) for tricriteria optimization problems can be seen as a hybrid of recursion and decomposition. It deals with upper bound vectors similar to other decomposition approaches. However, for a certain sequence of problems to be solved it keeps the bound on one of the objectives fixed as in a recursive method.

Bektaş (2018) presents a hybrid approach for multiobjective IPs that combines disjunctive constraints, inspired by Klein and Hannan (1982) and Sylva and Crema (2004), with decomposition. The disjunctive single-objective IPs use fewer variables than the original approaches, and the number of solver-calls (i.e., single-objective IPs solved) is reduced as compared to pure decomposition methods. However, the higher complexity of the individual solver calls still counteracts the speed-up obtained from solving fewer IPs.

5 Numerical study

For our numerical study we provide an implementation of the *defining point algorithm* (DPA) in C++. The source files as well as our own test files can be downloaded from https://github.com/kerstindaechert/DefiningPointAlgorithm. For the test files taken from other authors we refer to the respective urls. We compare our implementation to three state-of-the-art algorithms for which open source code is available and which are also implemented in `C++`. Moreover, all three methods invoke the commercial solver `CPLEX` to solve the single-objective scalarizations. Note that the respective implementations are of different types according to the classification shown in Sect. 4: disjoint decomposition (method "Epsilon" by Kirlik and Sayin 2014), recursive dimension reduction (method "AIRA" by Özlen et al. 2014) and a hybrid of disjunctive constraints and decomposition (method "DCM" by Boland et al. 2017a), supplemented by our "DPA" implementation which is a pure non-disjoint and non-redundant decomposition method.

We acknowledge that there are several other objective space methods that could be used for further comparisons, such as, for example, Bektaş (2018), Holzmann and Smith (2018), Tamby and Vanderpooten (2020). Our comparison are focused on the above methods Epsilon, AIRA and DCM for the following reasons: (1) `C++` implementations are available for these methods, and they can be combined with the same versions of single-objective solvers (`CPLEX` in our case). This leads to a fair comparison and avoids bias resulting from translations from other programming languages and/or re-implementations of existing code in `C++`. (2) The selected methods are representative in the sense that they cover different variants of objective space methods as discussed in Sect. 4 above. (3) In particular, Epsilon, the disjoint decomposition method of Kirlik and Sayin (2014), is used in a majority of all comparative studies on objective space methods. It can thus be seen as a general reference and may provide information on the relative performance w.r.t. other methods that are not included in this study.

In Sect. 5.1 we present our implementation in more detail. In Sect. 5.2 we state the benchmark algorithms with their current sources. We present the test instances in Sect. 5.3, followed by the discussion of the numerical results in Sect. 5.4.

5.1 Implementation

We implement the *defining point algorithm* (DPA) in `C++`. It is based on the original implementation of Klamroth et al. (2015) in `C` which has been designed to find and insert discrete sets of (nondominated) points. Hence, our main adjustment besides the translation to `C++` is to make it applicable to solve multiobjective integer programming problems. To do so we add an interface to the commercial solver CPLEX to solve ε-constraint scalarizations. Moreover, we implement the enhancement discussed in Sect. 3.

Algorithm 2 shows the pseudocode of our implementation. Note that it has the same structure as Algorithm 1, the Generic Scalarization-Based Algorithm. We read the MOIP problem from a textfile and transform it into minimization format if needed. In `ObtainBounds` we compute the ideal point z^I as well as a global upper bound z^M by minimizing or maximizing each objective function individually, respectively. The values of z^M incremented by 1 are used to define the upper bound u of the initial box. We initialize the list of boxes U by appending the initial box. As long as U is not empty, we select a box in `SelectBoxToRefine`. While it is possible to select any box here, in order to make use of Theorem 2, we select a box with minimal value u_i according to (6), where the index i has been specified by the user and is part of the Input of Algorithm 2.

Algorithm 2: Defining Point Algorithm with ε-constraint scalarization

```
Input   : MOIP problem, e-constraint index $i$
Output: Set of nondominated points $N$
1  $[z^I, z^M] \leftarrow$ ObtainBounds  // Compute ideal point and global upper bound
2  $u \leftarrow$ CreateInitialBox($z^M$)                         // Initialize $u$
3  $U \leftarrow \{u\}$                                   // Initialize list of boxes
4  $N \leftarrow \emptyset$                        // Initialize list of nondominated points
5  $s \leftarrow 1$                                              // Iteration counter
6  while $U \neq \emptyset$ do
7      $\bar{u} \leftarrow$ SelectBoxToRefine($U$)  // Get search zone with minimal value $u_i$
8      $(feasible, z^s) \leftarrow$ Solve($\bar{u}$)                  // Solve scalarization
9      if $feasible$ and $z_i^s < \bar{u}_i$ then
10         $N \leftarrow N \cup \{z^s\}$                     // save new nondominated point
11         $U \leftarrow$ UpdateList($U, z^s$)                  // update search region
12     else
13         $U \leftarrow U \backslash \{\bar{u}\}$     // remove explored search zone from search region
14     end
15     $s \leftarrow s + 1$
16 end
```

We then solve a scalarization, more precisely a two-stage or augmented ε-constraint method as presented in Sect. 2.2. We call the resulting variants DPA-TS and DPA-A, respectively. Note that we do not make use of the option of feeding the solver with a feasible starting solution. However, the defining point of the selected local upper bound $\bar{u}$ in component i is typically a feasible solution for $(P_i^\varepsilon(u))$ and $(P_i^{\varepsilon,\rho}(u))$ and might be the result of the scalarization when the box is empty. Hence, in case of feasibility, we additionally check whether $z_i^s < \bar{u}_i$. We append z^s only to the list N if the inequality is satisfied, otherwise we remove $\bar{u}$ from U.

In case a new nondominated point has been found, the list of local upper bounds is updated in `UpdateList`(U, z^s) which is basically a re-implementation of Algorithm 5 of Klamroth et al. (2015), however, without generating the i-child of $\bar{u}$ which yields an empty search zone according to Theorem 2. For the pseudocode and justification of this routine we refer to Klamroth et al. (2015).

5.2 Benchmark algorithms

In our numerical study we take into account recently proposed algorithms that offer an open-source implementation in `C++` and invoke the commercial solver CPLEX to solve scalarizations. To the best of our knowledge the following algorithms satisfy these requirements:

- Epsilon (Kirlik and Sayin 2014): http://home.ku.edu.tr/~moolibrary/

- AIRA (Özlen et al. 2014, Pettersson and Ozlen 2019): An implementation in C is available at https://bitbucket.org/melihozlen/moip_aira (last update 2017). A parallelized version of this code in C++ is available at https://github.com/WPettersson/moip_aira. It can also be used in a non-parallel fashion, hence, we used this implementation for our comparison.
- DCM (Boland et al. (2017a): http://www.eng.usf.edu/~hcharkhgard/

5.3 Instances

For problems with three to five objectives we use publicly available benchmark instances of knapsack, assignment and travelling salesman instances. For our tests with up to ten objectives we create new instances of knapsack and assignment problems.

5.3.1 Knapsack problem

We use knapsack instances from http://home.ku.edu.tr/~moolibrary/ which have been used as a benchmark in Kirlik and Sayin (2014) and are still used for comparison in recent publications. These instances include problems with three objectives and 10, 20, . . . , 100 items, problems with four objectives and 10, 20, 30, 40 items as well as problems with five objectives and 10 and 20 items. For each problem size there are 10 instances, so in total this set consists of 160 test files.

In order to create instances with six to ten criteria we used the test problem creator from https://github.com/wpettersson/ProblemGenerator.

5.3.2 Assignment problem

Assignment instances are also available at http://home.ku.edu.tr/~moolibrary/. These instances include problems with three objectives and 5, 10, . . . , 50 items. Again, for each problem size there are 10 instances.

We also created instances with more criteria with the help of the test problem creator from https://github.com/wpettersson/ProblemGenerator.

5.3.3 Travelling salesman problem

We use instances with three and four objectives from Pettersson and Ozlen (2019). Thereby, the problems with three objectives are available under the direct link https://figshare.com/articles/3_objective_problem_test_cases/4814695. The problems with four objectives are available here: https://figshare.com/articles/4_objective_problem_test_cases/4814698.

5.4 Numerical results

All algorithms are implemented in C++ and compiled for the Microsoft Windows 64 Bit operation system. For solving the optimization problems, the IBM CPLEX

Table 1 CPU times (absolute) on knapsack instances from Kirlik and Sayin (2014). If not all of the 10 problems in one instance could be solved within the given time, the number of solved problems is indicated in parentheses

p	n	$\lvert Z_N \rvert$	DPA-TS	DPA-A	Eps	DCM	AIRA
3	10	9.8	0.24	0.20	0.27	0.31	0.62
	20	38	1.14	0.87	1.27	1.41	1.82
	30	115.8	4.77	3.61	6.05	6.19	7.85
	40	311.2	17.13	12.35	22.49	26.27	28.72
	50	444.2	28.41	19.94	36.45	48.71	46.44
	60	917.1	72.54	50.85	98.14	166.48	118.86
	70	1643.4	161.86	114.58	232.77	534.98	265.45
	80	2295.8	249.08	174.51	354.34	911.59	393.98
	90	3107.8	410.20	293.55	665.13	1863.69	647.32
	100	5849	1035.90	758.72	2220.52	5753.29	1613.51
4	10	11.6	0.35	0.32	0.40	0.41	0.80
	20	136.8	9.95	8.77	32.24	14.37	22.38
	30	397.6	44.39	37.85	$385.32^{(9)}$	70.50	109.28
	40	1808.6	349.91	297.00	$629.49^{(2)}$	1091.37	1015.94
5	10	16.2	0.68	0.64	1.06	0.66	1.41
	20	161.2	29.71	28.76	$170.26^{(6)}$	44.11	87.69

Optimizer 12.10 solver is used. The comparisons were executed on an Intel Core i7–7500U @2.90 GHz with 16GB of RAM, which is a rather slow computer in comparison to most popular processors. However, it demonstrates that the described algorithms also run on personal laptops and do not require high-performance compute clusters. All algorithms generate a textfile as output that contains the generated nondominated points. We compare these output files to make sure that all algorithms work correctly.

Some of the compared algorithms measure the runtime when executing, others do not. Since we do not want to alter the algorithms, we measure runtime as the time between calling the main routine until its termination. In order to avoid excessive runtimes we use the following timeout rule for all knapsack and assignment instances: We measure the runtime of DPA-TS and use it as a reference. The maximum runtime for all other algorithms is at least 30 minutes and at most ten times the reference runtime. For the TSP instances we reduce the upper bound of the timeout to three (instead of ten) times the runtime of DPA-TS to obtain shorter overall running times. In order to obtain stable run times, we run every benchmark instance twice and compute the average runtime.

5.4.1 Knapsack instances from Kirlik and Sayin (2014)

We first study the knapsack instances with three to five objectives in Tables 1, 2, 3 and 4. Thereby, Table 1 shows CPU times, Table 2 the number of calls to CPLEX, thus, the number of solved IPs. For better comparison, the following two Tables 3 and 4

Table 2 Number of solved IPs (absolute) on knapsack instances from Kirlik and Sayin (2014). If not all of the 10 problems in one instance could be solved within the given time, the number of solved problems is indicated in parentheses

p	n	$\|Z_N\|$	DPA-TS	DPA-A	Eps	DCM	AIRA
3	10	9.8	34.4	24.6	40.3	18.4	52
	20	38	118.9	80.9	145.8	78.9	206.8
	30	115.8	351.9	236.1	446.9	258.2	654.8
	40	311.2	935.2	624	1202.7	727.6	1779.2
	50	444.2	1329	884.8	1727.8	1044.9	2567.7
	60	917.1	2742	1824.9	3571.8	2216.1	5319
	70	1643.4	4885.9	3242.5	6402.3	4022.5	9558.9
	80	2295.8	6816.6	4520.8	8976.7	5691.9	13,411.5
	90	3107.8	9248.4	6140.6	12,141.5	7671	18,148.3
	100	5849	17,379.4	11,530.4	22,930.2	14,620.5	34,315
4	10	11.6	59.8	48.2	75.5	24.9	125.3
	20	136.8	801.8	665	1211.5	514.1	2300.7
	30	397.6	2394.1	1996.5	$3423.6^{(9)}$	1666.9	7388.8
	40	1808.6	11,609.3	9800.7	$4849.5^{(2)}$	9044.6	37,130.3
5	10	16.2	142.2	126	199.8	43.7	415.2
	20	161.2	2033.6	1872.4	$1318.7^{(6)}$	1071.7	7571.2

Table 3 Relative CPU times (deviation from the fastest) on knapsack instances from Kirlik and Sayin (2014). If not all of the 10 problems in one instance could be solved within the given time, we do not indicate relative figures since this would be misleading

p	n	$\|Z_N\|$	DPA-TS	DPA-A	Eps	DCM	AIRA
3	10	9.8	1.19	1.01	1.29	1.54	3.30
	20	38	1.30	1	1.44	1.62	2.09
	30	115.8	1.33	1	1.67	1.70	2.16
	40	311.2	1.38	1	1.85	2.09	2.35
	50	444.2	1.42	1	1.79	2.35	2.30
	60	917.1	1.42	1	1.93	3.15	2.34
	70	1643.4	1.42	1	1.99	4.30	2.33
	80	2295.8	1.43	1	2.02	5.01	2.27
	90	3107.8	1.41	1	2.15	6.00	2.22
	100	5849	1.38	1	2.64	7.22	2.17
4	10	11.6	1.12	1.03	1.25	1.32	2.79
	20	136.8	1.15	1	2.62	1.45	2.45
	30	397.6	1.17	1	–	1.76	2.84
	40	1808.6	1.18	1	–	2.93	3.09
5	10	16.2	1.18	1.11	1.69	1.16	2.69
	20	161.2	1.14	1.05	–	1.32	2.90

show the same results but this time relative w.r.t. the best performing algorithm per problem, which means relative CPU time with respect to the fastest and the relative number of IP calls with respect to the one with the smallest number of IPs. Note that since we are showing average figures, it might happen that none of the algorithms reaches a value of 1 which means that it is not superior in all of the 10 problems.

Considering CPU times we note that DPA outperforms the three benchmark algorithms Epsilon, DCM and AIRA considerably, especially for large instance sizes. With growing problem size, the superiority of DPA becomes more pronounced. For example, for $p = 3$ and $n = 100$, AIRA requires a relative runtime of 2.17, EPS a relative runtime of 2.64 and DCM a relative runtime of even 7.22 compared to DPA-A. Expressed in absolute terms, this means a runtime of 758.72 s for DPA-A, 1613.51 s for AIRA, 2220.52 s for EPS and 5753.29 s for DCM.

Note that EPS ran out of the given time limit of ten times the runtime of DPA-TS in 13 of the 160 problems, all of them from instances 4–30, 4–40 and 5–20. To some extent this matches the results presented in Kirlik and Sayin (2014), where for 4–40 only five of the ten problems could be solved within a time limit of 25000 s of CPU time. AIRA and DCM solve all problems reliably.

Comparing DPA within its two variants, DPA-A is fastest in most of the instances. The two-stage variant DPA-TS requires between 1.12 up to 1.43 times longer CPU times. This goes in line with the fact that the two-stage variants solve one IP more per nondominated point compared to the augmented variant.

Comparing the number of CPLEX calls in Tables 2 and 4, DCM solves less IPs than all the other algorithms for the small instances. This is not surprising since DCM uses disjunctive programs to search multiple rectangular boxes at once. This reduces the number of CPLEX calls on the one hand, but yields more complicated models involving additional binary variables on the other hand. As can be seen from the CPU times, the strategy of solving less but more complicated problems does not pay off. Even if DPA-A solves much more IPs in some instances, it requires considerably less CPU time.

5.4.2 Assignment instances from Kirlik and Sayin (2014)

The results obtained for the benchmark assignment instances with three objectives are given in Tables 5, 6, 7, and 8. Again, we depict absolute CPU times and number of solved IPs as well as their relative conterparts. While the average number of nondominated points of these instances is similar to those of the knapsack instances, it is common knowledge that assignment problems are much harder to solve.

With respect to CPU time, the DPA algorithms perform best again. For instance 3–50 EPS requires 1.69 more runtime than DPA-A, followed by AIRA with a factor of 1.97 and DCM with a factor of 10.26. While DCM does not only consume much more CPU time than all other algorithms for this instance, it also does not find all nondominated points reliably. More precisely, for the ten problems contained in 3–50, it misses on average around 10% of the nondominated set.

With respect to the number of IPs, DPA-A solves the smallest number in all instances, followed by DCM, DPA-TS, EPS and AIRA, respectively.

Table 4 Relative number of solved IPs (deviation from the smallest) on knapsack instances from Kirlik and Sayin (2014). If not all of the 10 problems in one instance could be solved within the given time, we do not indicate relative figures since this would be misleading

p	n	$\lvert Z_N \rvert$	DPA-TS	DPA-A	Eps	DCM	AIRA
3	10	9.8	1.98	1.44	2.26	1	2.80
	20	38	1.57	1.07	1.91	1.02	2.69
	30	115.8	1.49	1	1.88	1.06	2.74
	40	311.2	1.50	1	1.92	1.15	2.84
	50	444.2	1.50	1	1.95	1.16	2.89
	60	917.1	1.50	1	1.95	1.21	2.91
	70	1643.4	1.51	1	1.97	1.24	2.95
	80	2295.8	1.51	1	1.98	1.25	2.96
	90	3107.8	1.51	1	1.98	1.25	2.95
	100	5849	1.51	1	1.99	1.27	2.97
4	10	11.6	2.59	2.10	3.09	1	4.82
	20	136.8	1.83	1.50	2.63	1	4.87
	30	397.6	1.51	1.25	–	1	4.56
	40	1809	1.35	1.13	–	1	4.22
5	10	16.2	3.43	3.02	4.64	1	9.30
	20	161.2	2.29	2.07	–	1	7.88

Table 5 CPU times (absolute) on assignment instances from Kirlik and Sayin (2014). For instance 3–50, DCM only found on average 90% of the nondominated points, marked by $*$

p	n	$\lvert Z_N \rvert$	DPA-TS	DPA-A	Eps	DCM	AIRA
3	5	14.1	0.33	0.25	0.30	0.30	0.67
	10	176.8	9.66	8.01	10.52	22.52	12.77
	15	674.9	72.69	56.26	87.46	155.50	108.05
	20	1860.5	283.41	209.34	346.41	680.95	416.30
	25	3567.8	681.26	492.85	819.14	1989.17	1017.63
	30	6181.3	1483.08	1071.81	1790.61	5319.32	2128.10
	35	8972.3	2605.24	1879.81	3127.74	11,445.82	3757.89
	40	14,679.7	4933.75	3526.23	5951.51	30,241.19	7037.86
	45	17,702.2	7039.59	5033.75	8401.81	48,333.81	9959.07
	50	24,916.8	11,776.82	8377.07	14,131.75	84,993.60$*$	16,420.50

5.4.3 Travelling salesman instances from Pettersson and Ozlen (2019)

As a third test bed we use travelling salesman instances with three and four objectives. Note that each problem size consists of five instances. Moreover, we set the timeout here to at most three times the runtime of DPA-TS (but again at least 30 min). The results are shown in Tables 9, 10, 11, and 12. The results are in line with the results for knapsack and assignment instances, and, thus, confirm the general behavior for three and four objectives: DPA-A is fastest in nearly all instances, followed by DPA-TS. The behavior of the three remaining algorithms depends on the problem size. For

Table 6 Number of solved IPs (absolute) on assignment instances from Kirlik and Sayin (2014). For instance 3–50, DCM only found on average 90% of the nondominated points, marked by $*$

p	n	$\lvert Z_N \rvert$	DPA-TS	DPA-A	Eps	DCM	AIRA
3	5	14.1	45.2	31.1	53	27.7	70.2
	10	176.8	472.6	295.8	572	352.2	841.9
	15	674.9	1743.9	1068.8	2128.2	1360.6	3178.4
	20	1860.5	4628.3	2767.9	5428.4	3676.3	8120.4
	25	3567.8	8643.9	5075.8	10,132.2	7028.8	15,322.9
	30	6181.3	14,753.4	8573.8	16,669.9	11,961.2	25,195.6
	35	8972.3	21,193.2	12,223.9	24,045.7	17,264.7	36,297.2
	40	14,679.7	34,035.8	19,368.1	38,363.8	27,933.3	57,969.9
	45	17,702.2	40,822.3	23,126.7	45,724.2	33,560.9	69,082.4
	50	24,916.8	56,876.1	31,965.6	62,968	41,067.4$*$	95,343

Table 7 Relative CPU times (deviation from the fastest) on assignment instances from Kirlik and Sayin (2014). For instance 3–50, DCM only found on average 90% of the nondominated points, marked by $*$

p	n	$\lvert Z_N \rvert$	DPA-TS	DPA-A	Eps	DCM	AIRA
3	5	14.1	1.3	1.04	1.25	1.21	2.87
	10	176.8	1.2	1	1.3	2.86	1.57
	15	674.9	1.35	1	1.66	3.24	1.98
	20	1860.5	1.30	1	1.56	2.82	1.92
	25	3567.8	1.39	1	1.66	3.97	2.06
	30	6181.3	1.39	1	1.67	4.94	1.99
	35	8972.3	1.39	1	1.67	6.05	2
	40	14,679.7	1.4	1	1.68	8.40	2
	45	17,702.2	1.40	1	1.68	9.51	1.99
	50	24,916.8	1.41	1	1.69	10.26$*$	1.97

Table 8 Relative number of solved IPs (deviation from the smallest) on assignment instances from Kirlik and Sayin (2014). For instance 3–50, DCM only found on average 90% of the nondominated points, marked by $*$

p	n	$\lvert Z_N \rvert$	DPA-TS	DPA-A	Eps	DCM	AIRA
3	5	14.1	1.67	1.16	1.94	1.00	2.53
	10	176.8	1.60	1	1.91	1.18	2.81
	15	674.9	1.63	1	1.99	1.27	2.98
	20	1860.5	1.67	1	1.95	1.32	2.92
	25	3567.8	1.70	1	1.99	1.38	3.01
	30	6181.3	1.72	1	1.95	1.40	2.94
	35	8972.3	1.73	1	1.97	1.41	2.97
	40	14,679.7	1.76	1	1.98	1.44	2.99
	45	17,702.2	1.77	1	1.98	1.45	2.99
	50	24,916.8	1.78	1	1.97	1.30*	2.99

Table 9 CPU times (absolute) on TSP instances from Pettersson and Ozlen (2019). For instance size 4–15 we only solve three of the five problems. If not all of the 5 (3) problems in one instance could be solved within the given time (at most three times the runtime of DPA-TS), the number of solved problems is indicated in parentheses

p	n	$\lvert Z_N \rvert$	DPA-TS	DPA-A	Eps	DCM	AIRA
3	10	250	24.24	16.15	27.22	34.00	33.72
	12	384	38.62	27.80	48.88	68.35	58.60
	15	1202	223.28	164.02	300.31	646.97	351.80
	20	3237.2	1071.85	772.78	1475.89	$-^{(0)}$	1662.71
	30	11,651.4	8324.52	6086.55	18,678.32	$-^{(0)}$	12,942.86
4	6	50.4	2.96	2.49	4.37	3.45	5.50
	8	253.8	31.32	27.27	145.16	58.26	68.94
	10	683.6	157.79	137.68	$535.70^{(2)}$	418.95	370.33
	12	3036	1348.79	1180.08	$-^{(1)}$	$-^{(2)}$	$1889.35^{(4)}$
	15(3)	8679.7	7493.48	6505.86	$-^{(0)}$	$-^{(0)}$	$-^{(1)}$

Table 10 Number of solved IPs (absolute) on TSP instances from Pettersson and Ozlen (2019). For instance size 4–15 we only solve three of the five problems. If not all of the 5 (3) problems in one instance could be solved within the given time (at most three times the runtime of DPA-TS), the number of solved problems is indicated in parentheses

p	n	$\lvert Z_N \rvert$	DPA-TS	DPA-A	Eps	DCM	AIRA
3	10	250	753.8	503.8	977.8	610	1448.2
	12	384	1151.8	767.8	1484.6	903	2200.8
	15	1202	3592.8	2390.8	4745	2945	7090.6
	20	3237.2	9640.4	6403.2	12,757.4	$-^{(0)}$	19,093.2
	30	11,651.4	34,554.8	22,903.2	45,737.4	$-^{(0)}$	68,502.6
4	6	50.4	276.2	225.8	406	156	758.2
	8	253.8	1567	1313.2	2522.6	1125.4	4927.6
	10	683.6	4455	3771.4	$3568^{(2)}$	3536	14,249.8
	12	3036	20,228	17,192	$-^{(1)}$	$-^{(2)}$	$42,134.5^{(4)}$
	15 (3)	8679.7	58,459	49,779.7	$-^{(0)}$	$-^{(0)}$	$107,469^{(1)}$

smaller problems, EPS is faster than DCM, which in turn is faster than AIRA. At some problem size, AIRA gets better than DCM. For large problems, AIRA outperforms DCM and EPS.

5.4.4 High-dimensional instances

Finally we investigate how the algorithms perform when applied to instances with up to 10 objectives. To the best of our knowledge, we are the first to present results for problems with more than six objectives by exact algorithms. Since the computational effort increases considerably with an increasing number of objectives, we restrict the

Table 11 Relative CPU times (deviation from the fastest) on TSP instances from Pettersson and Ozlen (2019). For instance size 4–15 we only solve three of the five problems. If not all of the 5 (3) problems in one instance could be solved within the given time (at most three times the runtime of DPA-TS), the number of solved problems is indicated in parentheses

p	n	$\|Z_N\|$	DPA-TS	DPA-A	Eps	DCM	AIRA
3	10	250	1.46	1	1.69	2.14	2.09
	12	384	1.39	1	1.75	2.42	2.10
	15	1202	1.36	1	1.83	3.89	2.14
	20	3237.2	1.39	1	1.90	$-^{(0)}$	2.15
	30	11,651.4	1.38	1	3.07	$-^{(0)}$	2.15
4	6	50.4	1.18	1.02	1.66	1.30	2.06
	8	253.8	1.14	1	4.37	2.05	2.48
	10	683.6	1.17	1	$-^{(2)}$	2.77	2.57
	12	3036	1.15	1	$-^{(1)}$	$-^{(2)}$	$2.83^{(4)}$
	15 (3)	8679.7	1.2	1	$-^{(0)}$	$-^{(0)}$	$-^{(1)}$

Table 12 Relative number of solved IPs (deviation from the smallest) on TSP instances from Pettersson and Ozlen (2019). For instance size 4–15 we only solve three of the five problems. If not all of the 5 (3) problems in one instance could be solved within the given time (at most three times the runtime of DPA-TS), the number of solved problems is indicated in parentheses

p	n	$\|Z_N\|$	DPA-TS	DPA-A	Eps	DCM	AIRA
3	10	250	1.50	1	1.93	1.21	2.86
	12	384	1.50	1	1.93	1.17	2.86
	15	1202	1.50	1	1.98	1.23	2.96
	20	3237.2	1.51	1	1.99	$-^{(0)}$	2.98
	30	11,651.4	1.51	1	2.00	$-^{(0)}$	2.99
4	6	50.4	1.97	1.6	2.72	1	4.88
	8	253.8	1.43	1.2	2.29	1	4.45
	10	683.6	1.40	1.17	$-^{(2)}$	1	4.29
	12	3036	1.27	1.07	$-^{(1)}$	$-^{(2)}$	$4.01^{(4)}$
	15 (3)	8679.7	1.2	1	$-^{(0)}$	$-^{(0)}$	$-^{(1)}$

number of knapsack items (variables) to at most 15. For the assignment instances with six to ten objectives we present results for 5 items, i.e., 25 variables. Moreover, we evaluate only five or three problems per instance.

The results of the knapsack instances are shown in Tables 13 and 14. First we state that both variants, DPA-TS and DPA-A, solve all instances reliably. Moreover, we observe that the main difference between both variants, the saving of $|Z_N|$ integer programs when using the augmented variant, looses importance with increasing number of objectives. This also translates to the CPU times. Considering the results of EPS we observe that it can only solve the smaller instances within the given time frame. While, e.g., DPA-TS solves instance 7–10 with on average 81.8 nondominated

Table 13 CPU times (absolute) on high-dimensional knapsack instances. The number of evaluated problems per instance size changes with p and n; it is reported in parentheses in the third column. The sign "–" means stopped by timeout

p	n	i	$\|Z_N\|$	DPA-TS	DPA-A	Eps	DCM	AIRA
6	5	(5)	6.6	0.29	0.26	0.34	0.21	0.52
	10	(5)	58.4	5.47	5.33	333.61	6.10	14.92
	15	(3)	86.3	15.09	14.55	–	21.75	49.56
7	5	(5)	10	0.64	0.63	3.12	0.43	1.75
	10	(5)	81.8	34.67	34.95	–	40.56	161.15
	15	(3)	182.7	305.55	304.82	–	$147.63^{(2)}$	$446.26^{(2)}$
8	5	(5)	10	0.82	0.85	23.47	0.47	2.40
	10	(5)	98.2	114.95	115.18	–	$124.46^{(2)}$	$455.71^{(3)}$
	15	(3)	65.3	161.28	161.39	–	358.64	$256.21^{(2)}$
9	5	(5)	10	1.05	1.19	332.54	0.51	3.55
	10	(5)	96.2	276.10	278.36	–	$620.74^{(4)}$	$1538.96^{(1)}$
	15	(3)	178.7	1923.23	1935.12	–	$7325.56^{(2)}$	–
10	5	(5)	14.2	1.56	1.70	–	0.49	3.99
	10	(5)	116	973.38	965.93	–	$2202.52^{(4)}$	–
	15	(3)	238.3	7328.30	6974.35	–	$86{,}402.6^{(2)}$	–

points within 34.67 s, EPS can not provide the nondominated set within half an hour. The smaller instances 6–5, 6–10, 7–5, 8–5 and 9–5 can be solved by EPS, but in all but one instance EPS is the slowest algorithm. AIRA also shows difficulties with the larger instances. In general, it performs better than EPS (except for instance 6–5) but worse than the DPA variants and DCM. DCM outperforms the DPA variants with respect to CPU times for the smaller instances 6–5, 7–5, 8–5, 9–5 and 10–5, for which the solution time is less than 1 s or slightly above. However, the DPA variants show their superiority in all larger instances. DCM can only partially solve these instances. The average figures shown in Table 13 only take the solved problems into account. Even for these, the average CPU times are, in general, higher than those of the DPA variants. Looking at the number of IPs, DCM solves significantly less IPs than both DPA variants in all high-dimensional instances. This underlines again that the number of IPs is not the main figure to look at but that the structure of these IPs, in particular the number of the involved integer variables, is decisive for the CPU time.

The results of the assignment instances are given in Tables 15 and 16. In general, the results are similar to the knapsack results. However, the DPA variants perform even better. Again, DPA-TS and DPA-A solve all instances reliably and are, apart from instances 4–5 and 5–5, clearly the fastest algorithms compared to the other three algorithms. However, different to the knapsack results, DCM is outperformed by the DPA variants also for most of the smaller instances. Besides, AIRA performs better for the assignment instances. It outperforms DCM in the larger instances 4–15, 4–20 and 5–10 with respect to CPU time, but is still inferior to the DPA variants.

Table 14 Number of solved IPs (absolute) on high-dimensional knapsack instances. The number of evaluated problems per instance size changes with p and n; it is reported in parentheses in the third column. The sign"–" means stopped by timeout

p	n	i	$\|Z_N\|$	DPA-TS	DPA-A	Eps	DCM	AIRA
6	5	(5)	6.6	74.4	67.8	90.4	15.4	177.8
	10	(5)	58.4	981.4	923	1531.4	326	4024.4
	15	(3)	86.3	1711.3	1625	–	751.7	8856
7	5	(5)	10	287.2	277.2	419.8	37.6	1107.8
	10	(5)	81.8	4280.6	4198.8	–	1308	22,772.4
	15	(3)	182.7	19,321	19,138.3	–	$2649^{(2)}$	$33{,}278^{(2)}$
8	5	(5)	10	370.4	360.4	504.4	33.8	1363
	10	(5)	98.2	13,765.2	13,667	–	$2878.5^{(2)}$	$35{,}266.5^{(3)}$
	15	(3)	65.3	11,293.3	11,228	–	1066	$33{,}272.5^{(2)}$
9	5	(5)	10	588.4	578.4	876.4	39.4	2748.2
	10	(5)	96.2	31,436.6	31,340.4	–	$8664.3^{(4)}$	$82{,}085^{(1)}$
	15	(3)	178.7	110,496.7	110,318	–	$27{,}285.5^{(2)}$	–
10	5	(5)	14.2	975.8	961.6	–	42.6	4329.6
	10	(5)	116	90,220.4	90,104.4	–	$12{,}537.2^{(4)}$	–
	15	(3)	238.3	364,318.7	364,080.3	–	$94{,}734.5^{(2)}$	–

Table 15 CPU times (absolute) on high-dimensional assignment instances. The number of evaluated problems per instance size changes with p and n; it is reported in parentheses in the third column. The sign "–" means stopped by timeout. For instance 4–20, DCM only found 28,869 of the 33,672 points, marked by $*$

p	n	i	$\|Z_N\|$	DPA-TS	DPA-A	Eps	DCM	AIRA
4	5	(10)	29.3	0.55	0.52	0.74	0.54	0.92
	10	(10)	971.2	184.85	159.74	330.61	315.78	366.41
	15	(4)	6787.5	2613.33	2253.98	5237.27	6707.09	5005.14
	20	(1)	33,672	29,365.56	26,813.56	59,572.00	86,732.04*	42,150.95
5	5	(10)	47.9	2.85	2.99	8.42	2.91	5.68
	10	(10)	3465.6	2240.03	2154.90	–	9146.58	6053.46
6	5	(10)	70.4	9.23	10.10	$571.77^{(9)}$	12.99	21.04
7	5	(5)	68.2	16.40	18.48	–	25.52	41.80
8	5	(5)	80.4	38.73	44.15	–	108.55	403.19
9	5	(5)	105.4	408.21	468.40	–	2030.70	–
10	5	(5)	106.8	986.12	1108.12	–	5514.01	–

6 Summary and further ideas

In this paper we combine the *defining point algorithm* (DPA) introduced in Klamroth et al. (2015) with suitable scalarization methods to obtain a new versatile approach to compute the nondominated set of multiobjective integer programming problems.

Table 16 Number of solved IPs (absolute) on high-dimensional assignment instances. The number of evaluated problems per instance size changes with p and n; it is reported in parentheses in the third column The sign "−" means stopped by timeout. For instance 4–20, DCM only found 28,869 of the 33,672 points, marked by $*$

p	n	i	$\|Z_N\|$	DPA-TS	DPA-A	Eps	DCM	AIRA
4	5	(10)	29.3	139.4	110.1	194.4	69.6	351
	10	(10)	971.2	4578.2	3608	6902.3	3279.2	13,714.3
	15	(4)	6787.5	33,351.75	26,655.25	45,366.75	22,864.75	91,014.25
	20	(1)	33,672	235,024	200,411	208,271	83,756*	424,282
5	5	(10)	47.9	476.8	428.9	783.7	223.9	1797.6
	10	(10)	3465.6	41,910.3	38,452.9	–	29,495.7	166,487.8
6	5	(10)	70.4	1476.6	1406.2	2079 $^{(9)}$	614.6	7130.7
7	5	(5)	68.2	2907.6	2839.4	–	893.6	16,007.4
8	5	(5)	80.4	7375	7294.6	–	2439.2	45,095.6
9	5	(5)	105.4	41,350.4	41,245	–	13,362.2	–
10	5	(5)	106.8	72,819.4	72,712.6	–	20,545.2	–

Our theoretical and numerical analysis provide evidence that DPA finds a competitive balance between the number of required solver calls on the one hand, and the numerical complexity of each individual solver call on the other hand. Indeed, we show that the number of solver calls can be bounded by a polynomial in the number of nondominated points in the worst case. At the same time, subproblems are kept simple by using basic ε-constraint or weighted Tchebychev scalarizations. Two variants of DPA are implemented in C++ and compared to available state-of-the-art open-source solvers. We demonstrate the clear superiority of DPA with respect to CPU time on common benchmark instances as well as newly generated instances with up to ten objectives. Setting a time limit of at most ten times the solution time of the fastest algorithm, respectively, only DPA solves all instances reliably.

We present our algorithm as versatile and modular in the sense that it can be combined with any scalarization. Our numerical study concentrates on the ε-constraint scalarization due to the advantages described in Sect. 3 when the goal is to find every efficient solution. A numerical comparison using an ε-constraint scalarization and additionally a weighted Tchebychev scalarization has been presented in Dächert and Klamroth (2015). The algorithm therein is structurally similar to the *defining point approach* used in this paper, so the findings can be transferred directly. It is shown that when the goal is to generate the entire nondominated set, the (augmented) ε-constraint scalarization is always superior to the weighted Tchebychev scalarization due to its reduced number of IPs. However, when the goal is to generate only a subset of the nondominated set, a so-called incomplete representation, then other scalarizations as, e.g., the weighted Tchebychev scalarization promise to be more useful. We refer to Doğan et al. (2022) for an implementation in the context of multi-objective mixed-integer linear programming (MOMILP) problems. An adaptation of DPA to other types of representations is a promising direction for future research. This topic is, however, beyond the scope of this paper and left for further research.

Our study also shows that with an increasing number of objectives, the computational time increases as well. Therefore, in the future, a parallel variant should be developed. Parallel variants based on other algorithms have already been proposed in Pettersson and Ozlen (2019) and Turgut et al. (2019). It remains to be studied how a *parallel defining point algorithm* competes with these approaches.

Acknowledgements We thank two anonymous referees for their valuable comments which helped to improve the paper. Kathrin Klamroth acknowledges financial support by the Deutsche Forschungsgemeinschaft, project number KL 1076/11-1.

Funding Open Access funding enabled and organized by Projekt DEAL.

Declarations

Conflict of interest We assure that there is no conflict of interest concerning the publication of this manuscript in MMOR. Kathrin Klamroth declares financial support by the Deutsche Forschungsgemeinschaft, project number KL 1076/11-1. The other two authors have not received specific funding. All sources of data that have been used for this work are properly declared and referenced.

References

Aneja YP, Nair KPK (1979) Bicriteria transportation problem. Mangement Science 25:73–78

Bektaş T (2018) Disjunctive programming for multiobjective discrete optimisation. INFORMS J Comput 30(4):625–633

Boissonnat JD, Sharir M, Tagansky B et al (1998) Voronoi diagrams in higher dimensions under certain polyhedral distance functions. Discrete Comput Geometry 19:485–519

Boland N, Charkhgard H, Savelsbergh M (2016) The L-shape search method for triobjective integer programming. Math Program Comput 8:217–251

Boland N, Charkhgard H, Savelsbergh M (2017a) A new method for optimizing a linear function over the efficient set of a multiobjective integer program. Eur J Oper Res 260(3):904–919

Boland N, Charkhgard H, Savelsbergh M (2017b) The quadrant shrinking method: a simple and efficient algorithm for solving tri-objective integer programs. Eur J Oper Res 260(3):873–885

Chalmet L, Lemonidis L, Elzinga D (1986) An algorithm for the bi-criterion integer programming problem. Eur J Oper Res 25:292–300

Dächert K, Klamroth K (2015) A linear bound on the number of scalarizations needed to solve discrete tricriteria optimization problems. J Global Optim 61(4):643–676

Dächert K, Gorski J, Klamroth K (2012) An augmented weighted Tchebycheff method with adaptively chosen parameters for discrete bicriteria optimization problems. Comput Oper Res 39:2929–2943

Dächert K, Klamroth K, Lacour R et al (2017) Efficient computation of the search region in multi-objective optimization. Eur J Oper Res 260(3):841–855

Dhaenens C, Lemesre J, Talbi EG (2010) K-PPM: a new exact method to solve multi-objective combinatorial optimization problems. Eur J Oper Res 200:45–53

Doğan I, Lokman B, Köksalan M (2022) Representing the nondominated set in multi-objective mixed-integer programs. Eur J Oper Res 296:804–818

Ehrgott M (2005) Multicriteria optimization. Springer, Berlin

Ehrgott M (2006) A discussion of scalarization techniques for multiple objective integer programming. Ann Oper Res 147:343–360

Ehrgott M, Ruzika S (2008) Improved ε-constraint method for multiobjective programming. J Optim Theory Appl 138:375–396

Ehrgott M, Tenfelde-Podehl D (2003) Computation of ideal and Nadir values and implications for their use in MCDM methods. Eur J Oper Res 151:119–139

Figueira et al. (2017) Easy to say they're hard, but hard to see they're easy - toward a categorization of tractable multiobjective combinatorial optimization problems. J Multi-Criteria Decis Anal 24:82–98

Holzmann T, Smith J (2018) Solving discrete multi-objective optimization problems using modified augmented weighted Tchebychev scalarizations. Eur J Oper Res 271:436–449

Joswig M, Loho G (2020) Monomial tropical cones for multicriteria optimization. SIAM J Discrete Math 34:1172–1191

Kaplan H, Rubin N, Sharir M et al (2008) Efficient colored orthogonal range counting. SIAM J Comput 38:982–1011

Kirlik G, Sayın S (2014) A new algorithm for generating all nondominated solutions of multiobjective discrete optimization problems. Eur J Oper Res 232:479–488

Klamroth K, Lacour R, Vanderpooten D (2015) On the representation of the search region in multi-objective optimization. Eur J Oper Res 245(3):767–778

Klein D, Hannan E (1982) An algorithm for the multiple objective integer linear programming problem. Eur J Oper Res 9:378–385

Laumanns M, Thiele L, Zitzler E (2005) An adaptive scheme to generate the pareto front based on the epsilon-constraint method. In: Branke J, Deb K, Miettinen K, et al. (eds) Practical approaches to multi-objective optimization. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, Dagstuhl, Germany, no. 04461 in Dagstuhl Seminar Proceedings, http://drops.dagstuhl.de/opus/volltexte/2005/246

Laumanns M, Thiele L, Zitzler E (2006) An efficient, adaptive parameter variation scheme for metaheuristics based on the epsilon-constraint method. Eur J Oper Res 169:932–942

Lokman B, Köksalan M (2013) Finding all nondominated points of multi-objective integer programs. J Global Optim 57:347–365

Miettinen K (1999) Nonlinear multiobjective optimization. Kluwer Academic Publishers, Boston

Nemhauser GL, Wolsey LA (1999) Integer and combinatorial optimization. Wiley

Özlen M, Azizoğlu M (2009) Multi-objective integer programming: a general approach for generating all non-dominated solutions. Eur J Oper Res 199:25–35

Özlen M, Burton BA, MacRae CAG (2014) Multi-objective integer programming: an improved recursive algorithm. J Optim Theory Appl 160(2):470–482

Pettersson W, Ozlen M (2019) Multi-objective integer programming: Synergistic parallel approaches. INFORMS J Comput

Przybylski A, Gandibleux X, Ehrgott M (2010) A two phase method for multi-objective integer programming and its application to the assignment problem with three objectives. Discrete Optim 7:149–165

Ralphs T, Saltzman M, Wiecek MM (2006) An improved algorithm for solving biobjective integer programs. Ann Oper Res 147:43–70

Sylva J, Crema A (2004) A method for finding the set of non-dominated vectors for multiple objective integer linear programs. Eur J Oper Res 158:46–55

Sylva J, Crema A (2008) Enumerating the set of non-dominated vectors in multiple objective integer linear programming. RAIRO-Oper Res 42(3):371–387

Tamby S (2018) Approches génériques pour la résolution de problèmes d'optimisation discrète multiobjectif. PhD thesis, Université Paris-Dauphine, in French

Tamby S, Vanderpooten D (2020) Enumeration of the nondominated set of multiobjective discrete optimization problems. INFORMS J Comput

Tenfelde-Podehl D (2003) A recursive algorithm for multiobjective combinatorial optimization problems with Q criteria. Institut für Mathematik, Technische Universität Graz, Tech. rep

Turgut O, Dalkiran E, Murat A (2019) An exact parallel objective space decomposition algorithm for solving multiobjective integer programming problems. J Global Optim 75:35–62