An exact algorithm for biobjective integer programming problems

Saliha Ferda Doğan⋆†  Özlem Karsu ‡  Firdevs Ulus §

May 21, 2019

Abstract

We propose an exact algorithm for solving biobjective integer programming problems, which arise in various applications of operations research. The algorithm is based on solving Pascoletti-Serafini scalarizations to search specified regions (boxes) in the objective space and returns the set of nondominated points. We develop variants of the algorithm, where the choice of the scalarization model parameters differ; and demonstrate their performance through computational experiments both as exact algorithms and as solution approaches under time restriction. The results of our experiments show the satisfactory behaviour of our algorithm, especially with respect to the number of mixed integer programming problems solved compared to an existing approach. The experiments also demonstrate that different variants have advantages in different aspects: while some variants are quicker in finding the whole set of nondominated solutions, other variants return good-quality solutions in terms of representativeness when run under time restriction.

Keywords: Biobjective integer programming, Pascoletti-Serafini scalarization, Algorithms.

1 Introduction

In many operations research applications such as scheduling, task assignment and transportation, the underlying problem is an integer programming problem. Moreover, a vast amount of these problems require more than a single criterion to be considered, leading to biobjective (multiobjective) integer programming problems.

In this study, we focus on biobjective integer programming problems (BOIP) and propose an algorithm that returns the whole set of nondominated points of these problems. There are a number of solution approaches that have been designed for BOIP in the literature, most of which explore the objective (criterion) space by repetitively solving single objective optimization problems related to the BOIP, called scalarization problems (or simply, scalarizations). A scalarization is formulated by means of a real-valued scalarizing function of the objective functions of the BOIP, auxiliary scalar or vector variables and/or parameters (\[\text{[7]}\]).

⋆All authors contributed equally to this work. All are from Bilkent University, Department of Industrial Engineering, Ankara, 06800 Turkey
†ferda.dogan@bilkent.edu.tr
‡ozlemkarsu@bilkent.edu.tr
§firdevs@bilkent.com.tr
There are several scalarizations proposed in the literature. The widely-used ones are the weighted sum scalarization ([9, 17]), the \( \epsilon \)-constraint scalarization ([8]) and the (weighted) Chebyshev scalarization ([2, 15]). Most of the current algorithms in the literature solve these scalarizations or their modifications repetitively to find the set of nondominated solutions. Commonly used ones are the perpendicular search and the \( \epsilon \)-constraint algorithm, which are based on weighted sum scalarization and \( \epsilon \)-constraint scalarization, respectively ([4, 5, 8]). Examples of algorithms using weighted Chebychev scalarizations are proposed in [12] and [14], where a modified version of the scalarization is used. There are also two-phase algorithms, which generate supported nondominated points in the first phase and find the unsupported nondominated points by exploring the triangles defined by two consecutive supported nondominated points in the second phase ([11, 16]). Recently, the balanced box algorithm is proposed in [1] and a two-stage algorithm which combines the balanced box and \( \epsilon \)-constraint algorithms is discussed in [6].

We propose an exact solution algorithm that finds the whole set of nondominated solutions to biobjective integer programming problems. The algorithm is based on Pascoletti-Serafini scalarization ([10]). We adapt this scalarization model for biobjective integer programming settings and develop different variants of the algorithm. We compare these variants with respect to number of (mixed) integer programming problems solved and solution time. We also test the performances of the algorithm under time limit and report on the representativeness of the obtained solution sets using the (scaled) coverage error ([3, 13]).

The structure of the paper is as follows. In Section 2 we give the preliminaries and the problem definition. In Sections 3 and 4 we explain the base algorithm and its variants, respectively. We test the performances of the algorithm and report the results of our experiments in Section 5. We conclude our discussion in Section 6.

2 Preliminaries and problem definition

A general bi-objective integer programming problem is formulated as

\[
\text{“min”} \{ \, z(x) = (z_1(x), z_2(x))^T \mid x \in \mathcal{X} \subseteq \mathbb{Z}^n \} , \quad (P)
\]

where \( z_i(\cdot), \, i = 1, 2 \) are integer-valued objective functions. The set \( \mathcal{X} \) represents the feasible set in the decision space and the set \( \mathcal{Z} := \{ z(x) \mid x \in \mathcal{X} \} \) represents the feasible set in the objective space.

Throughout the paper we will use the following notation for vector inequalities:

\[
\begin{align*}
    z(x') & \leq z(x) : \iff z_i(x') \leq z_i(x) \text{ for } i \in \{1, 2\}; \\
    z(x') & < z(x) : \iff z(x') \leq z(x) \text{ and } z(x') \neq z(x); \\
    z(x') & < z(x) : \iff z_i(x') < z_i(x) \text{ for } i \in \{1, 2\}.
\end{align*}
\]

**Definition 2.1.** \( z(x') \in \mathcal{Z} \) dominates (strictly dominates) \( z(x) \in \mathcal{Z} \) if \( z(x') \leq z(x) \) and \( z(x') < z(x) \). If there exists no \( x' \in \mathcal{X} \) such that \( z(x') \) dominates (strictly dominates) \( z(x) \), then \( z(x) \) is nondominated (weakly nondominated) and \( x \) is efficient (weakly efficient).

The set of all nondominated vectors is denoted by \( \mathcal{N} \). The ideal and nadir points of problem \((P)\) are as follows:

\[
    s^0 := \left( \min_{x \in \mathcal{X}} z_1(x), \min_{x \in \mathcal{X}} z_2(x) \right)^T, \quad u^0 := \left( \max_{z \in \mathcal{N}} z_1, \max_{z \in \mathcal{N}} z_2 \right)^T.
\]
A lexicographic optimization problem with two objective functions is given by

\[
\text{lexmin}\{z_i(x), z_j(x) \mid x \in \mathcal{X}\}, \quad (1)
\]

where \(i, j \in \{1, 2\}\) and \(i \neq j\). Solving (1) means solving the following two (single objective) optimization problems: First, \(\min\{z_i(x) \mid x \in \mathcal{X}\}\); and given an optimal solution \(x'\) of the first model, \(\min\{z_j(x) \mid x \in \mathcal{X}, z_i(x) = z_i(x')\}\). Solving a lexicographic optimization yields an efficient solution.

In general, scalarization models are solved in order to find (weakly) efficient solutions. Throughout, Pascoletti-Serafini scalarization, which employs two parameters, a reference point \(s \in \mathbb{R}^2\) and a direction \(d \in \mathbb{R}^2_+ \setminus \{0\}\) is employed. The model is as follows:

\[
\min\{\rho \mid x \in \mathcal{X}, z(x) \leq s + \rho d, \rho \in \mathbb{R}\}. \quad (2)
\]

**Lemma 2.2** ([10]). If \((x^*, \rho^*)\) is an optimal solution of (2) for some \(s \in \mathbb{R}^2\) and \(d \in \mathbb{R}^2_+ \setminus \{0\}\), then \(x^*\) is weakly efficient.

**Lemma 2.3.** If \((x^*, \rho^*)\) is an optimal solution of (2) for some \(s \in \mathbb{R}^2\) and \(d \in \mathbb{R}^2_+ \setminus \{0\}\) then \(y^* := s + \rho^* d\) and \(z(x^*)\) are equal in at least one component.

**Proof.** Proof Assume to the contrary that \(z(x^*)\) and \(y^*\) are different in both components. That is, \(z(x^*) < y^*\). Hence there exists \(\bar{\rho} < \rho^*\) such that \(z(x^*) \leq s + \bar{\rho} d\), which contradicts the optimality of \((x^*, \rho^*)\). \(\square\)

When a subset \(\tilde{\mathcal{N}}\) of \(\mathcal{N}\) is found through an algorithm or a procedure, in order to measure how well \(\tilde{\mathcal{N}}\) represents the set of all nondominated points \((\mathcal{N})\), it is possible to use the ‘coverage error’ that is introduced by [13]. Similar measures are used in the literature to measure representativeness, an example is the coverage gap measure used recently in [3]. Here we provide the definition of coverage error for the special case where Chebyshev metric is used. We also introduce the scaled version as in [3].

**Definition 2.4.** Let \(\tilde{\mathcal{N}} \subseteq \mathcal{N}\) be a representative subset. The coverage error of \(\tilde{\mathcal{N}}\) with respect to \(n \in \mathcal{N}\) is

\[
CE(\tilde{\mathcal{N}}, n) := \min_{\tilde{n} \in \tilde{\mathcal{N}}} \left(\max\{|n_1 - \tilde{n}_1|, |n_2 - \tilde{n}_2|\}\right),
\]

the coverage error of \(\tilde{\mathcal{N}}\) is

\[
CE(\tilde{\mathcal{N}}) = \max_{n \in \mathcal{N}} CE(\tilde{\mathcal{N}}, n)
\]

and the scaled coverage error of \(\tilde{\mathcal{N}}\) is

\[
SCE(\tilde{\mathcal{N}}) = \frac{CE(\tilde{\mathcal{N}})}{\max\{u_0^0 - s_1^0, u_2^0 - s_2^0\}},
\]

where \(u^0\) and \(s^0\) are the nadir and the ideal points, respectively.
3 The algorithm

Throughout the algorithm the search regions in the objective space are referred to as boxes. A box is defined by three points in the criterion space, namely the starting point $s$, the nondominated point $t$ which defines the first component of the starting point and the nondominated point $p$ which defines the second component of the starting point, and denoted as follows

$$b(s, p, t) := \{ y \in \mathbb{R}^2 \mid s_1 \leq y_1 \leq p_1, \ s_2 \leq y_2 \leq t_2 \}.$$

Note that it is possible to define the box using only $p$ and $t$. However, we keep the starting point $s$ in the definition as it is used in the scalarization models.

The general idea of the algorithm can be described as follows. At the beginning, two sets namely $\mathcal{N}$ and $\mathcal{B}$, are defined to denote the set of nondominated points and boxes to be investigated, respectively. For initialization, two corner points of the nondominated set are found by solving $\text{lexmin}\{z_1(x), z_2(x)\mid x \in \mathcal{X}\}$ and $\text{lexmin}\{z_2(x), z_1(x)\mid x \in \mathcal{X}\}$. Let the optimal objective function vectors of these models be $t^0_0$, $p^0_0$, respectively. We initialize $\mathcal{N}$ as $\{t^0_0, p^0_0\}$ and $\mathcal{B}$ as $\{b(s^0_0, p^0_0, t^0_0)\}$, where $s^0_0$ is the ideal point. Clearly, the initial box includes all nondominated points. See Figure 1 for the illustration of the initial region.

![Figure 1: Initial box](image)

At each iteration, the algorithm searches one box from set $\mathcal{B}$ to find a (weakly) nondominated point by solving a Pascoletti-Serafini scalarization. In order to ensure finding a nondominated point, an (two) extra model(s) is (are) solved and the obtained nondominated point(s) is (are) added to $\mathcal{N}$. Then, the explored box is discarded and if at least a new nondominated point is found, two new boxes are added to $\mathcal{B}$ to be searched in the next iterations. The algorithm continues until there are no boxes to explore. The pseudocode of the algorithm is given by Algorithm 1.

At an arbitrary iteration, a box $b := b(s^b, p^b, t^b)$ from set $\mathcal{B}$ is selected and the following optimization problem is solved to search the box

$$\min \{ \rho \mid x \in \mathcal{X}, \ z(x) \leq s^b + \rho d, \ z_1(x) \leq p^b_1 - \epsilon, \ z_2(x) \leq t^b_2 - \epsilon \}, \quad (R(b, d))$$

where $d$ a is direction vector set to $d = (1, 1)^T$ and $0 < \epsilon < 1$. This is a slightly modified Pascoletti-Serafini model. The last two constraints are added to prevent finding the nondominated points $p^b$ and $t^b$, which are already found in the previous iterations. If this problem is infeasible, then there is no nondominated point other than $p^b$ and $t^b$ in the box. Otherwise, let the optimal solution of $(R(b, d))$ be $(\rho^b, x^b)$ and the corresponding (weakly) nondominated
Figure 2: \((P_1(x^b))\) is solved, \(n^1\) is found as a nondominated point and \(n^2\) is set to \(n^b\).

Figure 3: \((P_2(x^b))\) is solved, \(n^2\) is found as nondominated point and \(n^1\) is set to \(n^b\).

Figure 4: \((P_1(x^b))\) and \((P_2(x^b))\) are solved, \(n^1\) and \(n^2\) are found as nondominated points.

point be \(n^b = z(x^b)\), see Lemma 2.2. Note that \(\rho^b\) is the step size and defines the point \(y^b := s^b + \rho^bd\), which has at least one common component with \(n^b\), see Lemma 2.3. Since the scalarization only guarantees that \(n^b\) is weakly nondominated, the following problem(s) is (are) solved to ensure that a nondominated point is found. If the first components of \(y^b\) and \(n^b\) are equal \((n^b_1 = y^b_1)\) then,

\[
\min \{ z_2(x) | x \in \mathcal{X}, \ z_1(x) = z_1(x^b) \} \quad (P_1(x^b))
\]

is solved and, if the second components are equal \((n^b_2 = y^b_2)\) then,

\[
\min \{ z_1(x) | x \in \mathcal{X}, \ z_2(x) = z_2(x^b) \} \quad (P_2(x^b))
\]

is solved. Notice that it is possible to have \(y^b = n^b\) and in this case, both problems are solved. Let the solutions of \((P_1(x^b))\) and \((P_2(x^b))\) be \(x^1\) and \(x^2\), respectively and \(n^1 := z(x^1)\) and \(n^2 := z(x^2)\) be the corresponding points in the criterion space. If only \((P_1(x^b))\) is solved, then \(n^2\) is set to \(n^b\) and symmetrically, if only \((P_2(x^b))\) is solved \(n^1\) is set to \(n^b\) (to be used in partitioning) (see lines 8-17 in Algorithm 1). Then, \(\mathcal{N}\) is updated accordingly (lines 18-23).

If both \((P_1(x^b))\) and \((P_2(x^b))\) are solved, it is possible to find two nondominated points \(n^1\) and \(n^2\) in the same iteration. In this case, both \(n^1\) and \(n^2\) are added to \(\mathcal{N}\). See Figures 2-4 for illustrations of these cases.

As for any (weakly) nondominated point \(n\), the dominated region \(\{y \in \mathbb{R}^2 | n \leq y\}\) and the dominating region \(\{y \in \mathbb{R}^2 | y \leq n\} \cup \{y \in \mathbb{R}^2 | y < n\}\) can not contain any nondominated
points, the current box \(b(b^i, p^i, t^i)\) is split into two boxes using \(n^1\) and \(n^2\). More specifically, the first box is formed as \(b(s^1, p^i, n^1)\), where \(s^1 := (n^1_1, n^1_2)^T\) and the second box is formed as \(b(s^2, n^2, t^i)\), where \(s^2 := (t^i_1, n^2_2)^T\). See Figures 3-4 for the illustrations of newly formed boxes for different cases.

Finally, we avoid searching regions which can not have any new nondominated points, by taking the advantage of the integrality of the problem \((P)\), and the structure of a box. The boxes which do not satisfy \(p^i - s^1_1 > 1\) and \(t^i_2 - s^2_2 > 1\) are eliminated since they can not include any nondominated points other than \(p^i\) and \(t^i\). After new boxes are defined and their sizes are checked to make sure that they can include nondominated points, they are added to set \(B\) to be searched in the next iterations. Then, the searched box \(b(s^*, p^*, t^*)\) is removed from the set \(B\) (lines 24-29). The algorithm repeats the steps which are introduced above until there is no box in \(B\).

The algorithm works correctly and returns the set of all nondominated points after finitely many iterations. These are shown by the following two propositions.

**Proposition 3.1.** Algorithm 1 works correctly: It returns the set of all nondominated points.

*Proof.* The points that are added to set \(\mathcal{N}\) are guaranteed to be nondominated. Indeed, \(\{R(b, d)\}\) is a Pascoletti-Serafini scalarization with box contraints and by Lemma 2.2 it returns a weakly efficient solution. By solving \((P_1(x^b))\) and/or \((P_2(x^b))\), finding an efficient solution is guaranteed. Moreover, by the structure of defining the new boxes, it is guaranteed that the set of all boxes to be searched \((\mathcal{B})\) include all the remaining (if any) nondominated points at any time through the algorithm.

**Proposition 3.2.** Algorithm 1 solves \((3N + C - 3C_2 - E - 1)\) integer programs, where \(N = |\mathcal{N}|\) is the number of nondominated points, \(C\) is the number of cases where \((y^b = n^b)\), \(C_2\) is the number of sub-cases that two nondominated points are found and \(E\) is the number of eliminated boxes using the elimination rule.

*Proof.* The following expression, parts (a) – (g) of which will be explained in detail, shows the number of models solved:

\[
\begin{align*}
(a) & \quad (4) + (1) \\
(b) & \quad (2C_2) \\
(c) & \quad 2(N - 2 - 2C_2) \\
(d) & \quad (N - 2) \\
(e) & \quad (C - C_2) \\
(f) & \quad E - 1 \\
(g) & \quad E
\end{align*}
\]

At the beginning of Algorithm 1, two lexicographical minimization problems are solved to find \(t^0\) and \(p^0\) (a) and one \(\{R(b, d)\}\) problem is solved to search the initial box (b). \(2C_2\) points are found in \(C_2\) number of cases (\(y^b = n^b\) and two solutions are found), each of these points lead to a new box, hence a new \(\{R(b, d)\}\) model (c). For the rest of the nondominated points, \((N - 2C_2 - 2)\), each point results in two new boxes (and hence two \(\{R(b, d)\}\) models to be solved) (d). As for the \((P_1(x^b))\) models: \(N\) \(-\) 2 points are found by solving a single second stage model (either \((P_1(x^b))\) or \((P_2(x^b))\) (e). Moreover, when \(y^b = n^b\) and only a single nondominated point is found (in \(C - C_2\) number of cases), we solve an extra \((P_1(x^b))\) or \((P_2(x^b))\), which does not yield a new point (f). Finally, \(E\) boxes are eliminated, avoiding the \(\{R(b, d)\}\) models that would otherwise have been solved (g).

4 The variants

The algorithm described in Section 3 can be modified in different ways. First of all, in each iteration the direction parameter \(d\) can be chosen according to the current box instead of fixing
Algorithm 1: The Proposed Algorithm for BOIP

**Input**: Problem \( P \)

**Output**: The set of all nondominated solutions \( \mathcal{N} \)

### 1 Initializations

1. **Initialization 1** \( (I_1) \)
   \[ d = (1, 1)^T, \epsilon < 1 \]

2. **Initialization 2** \( (I_2) \)
   Solve \( \text{lexmin}\{z_1(x), z_2(x) | x \in \mathcal{X}\} \)

3. **Initialization 3** \( (I_3) \)
   Solve \( \text{lexmin}\{z_2(x), z_1(x) | x \in \mathcal{X}\} \)

4. **Initialization 4** \( (I_4) \)
   \[ \mathcal{N} = \{t^0, p^0\}, s^0 = (t^0, p^0)^T, \mathcal{B} = \{b(s^0, p^0, t^0)\} \]

### 2 MainLoop

while \( \mathcal{B} \) is not empty do

1. **MainLoop 1** \( (R(b, d)) \)
   Let \( b(s^b, p^b, t^b) \in \mathcal{B} \) and solve \( (R(b, d)) \)

2. **MainLoop 2** \( (R(b, d)) \)
   if \( (R(b, d)) \) is feasible then
   \[ y^b = s^b + \rho^b d \]
   \[ n^b = z(x^b) \]
   if \( y^b_1 = n^b_1 \) then
   Solve \( (P_1(x^b)) \). Let \( x^1 \) be an optimal solution.
   \[ n^1 = z(x^1) \]
   else
   \[ n^1 = n^b \]
   if \( y^b_2 = n^b_2 \) then
   Solve \( (P_2(x^b)) \). Let \( x^2 \) be an optimal solution.
   \[ n^2 = z(x^2) \]
   else
   \[ n^2 = n^b \]
   if \( n^2_2 < n^b_2 \) then
   \[ \mathcal{N} \leftarrow \mathcal{N} \cup \{n^1\} \]
   if \( n^2_1 < n^b_1 \) then
   \[ \mathcal{N} \leftarrow \mathcal{N} \cup \{n^2\} \]
   if \( n^2_1 \geq n^b_1 \) and \( n^2_2 \geq n^b_2 \) then
   \[ \mathcal{N} \leftarrow \mathcal{N} \cup \{n^b\} \]
   \[ s^1 = (n^1_1, p^b_2)^T \]
   \[ s^2 = (t^b_1, n^b_2)^T \]
   if \( p^b_1 - s^1_1 > 1 \) and \( n^1_2 - s^1_2 > 1 \) then
   \[ \mathcal{B} \leftarrow \mathcal{B} \cup \{b(s^1, p^b_1, n^1)\} \]
   if \( n^2_1 - s^2_1 > 1 \) and \( t^b_2 - s^2_2 > 1 \) then
   \[ \mathcal{B} \leftarrow \mathcal{B} \cup \{b(s^2, n^2_2, t^b)\} \]
   \[ \mathcal{B} \leftarrow \mathcal{B} \backslash \{b(s^b, p^b, t^b)\} \]
it to \((1, 1)^T\) (Fixed). We consider two alternatives: to choose \(d\) as the diagonal direction of the current box, \(d = (p^b_1 - s^b_1, t^b_2 - s^b_2)^T\) (Changing) and to choose \(d\) as the direction starting from \(s^b\) towards the nadir point, \(d = (p^0_1 - s^b_1, t^0_2 - s^b_2)^T\) (Nadir). Secondly, the new search regions added to \(B\) in each iteration can be chosen differently. In addition to the base version that is described in Section 3 we consider employing \(y^b = s^b + \rho^b d\) in defining the new regions. Accordingly, we use \(y^b\) instead of \(n^b\) in order to define a new box if it yields a smaller region than the base version, see Figures 5 and 6. This is done by replacing lines 12 and 17 of Algorithm 1 with \(n^1 = y^b\) and \(n^2 = y^b\), respectively. Notice that since the corners of the newly formed boxes are not necessarily integer valued, one also needs to change the elimination rule slightly. More specifically, for this variant the strict inequalities on lines 26 and 28 of Algorithm 1 are replaced by greater than or equal to signs.

The six variants can be seen in Table 1 The base version that is described in Section 3 is denoted by FDN.

Table 1: The variants of the algorithm

| Variants     | Fixed     | Changing  | Nadir     |
|--------------|-----------|-----------|-----------|
|              | \(d = (1, 1)^T\) | \(d = (p^b_1 - s^b_1, t^b_2 - s^b_2)^T\) | \(d = (p^0_1 - s^b_1, t^0_2 - s^b_2)^T\) |
| Using \(n^b\) (always) | FDN       | CDN       | NDN       |
| Using \(y^b\) (if smaller) | FDY       | CDY       | NDY       |

Figure 5: Forming new boxes using \(n^b\)  
Figure 6: Forming new boxes using \(y^b\)

5 Computational results

We examine the efficiency of the algorithms by solving knapsack and assignment problem instances which are used in [1]. Both problem types contain four classes, A, B, C, D each with five instances. The first set consists of biobjective Knapsack Problem (KP) instances with 375, 500, 625 and 750 variables. The second set consists of biobjective Assignment Problem (AP) instances with 200 \(\times\) 200 and 300 \(\times\) 300 binary variables.

1The instances are available at http://hdl.handle.net/1959.13/1036183
The algorithms are coded in C++ and all mixed integer programming models are solved using CPLEX 12.6. Only a single thread is used. All of the instances are run in a computer with Intel Xeon CPU E5-1650 3.6 GHz processor and 32 GB RAM. Computation times are given in central processing unit (CPU) seconds.

We first conduct preliminary experiments on type A knapsack and assignment instances in order to compare the performances of the algorithm variants. In Table 2, we report the average values for the number of nondominated points ($N_{avg}$), the number of all (mixed) integer programming problems solved, the solution time (in CPU seconds), the number of ($R(b,d)$) models solved, average time for solving one ($R(b,d)$) model, average time for solving one ($P_i(x^b)$) model, $C$, $C_2$ and $E$, see Proposition 3.2.

Overall, we see that partitioning a box using nondominated points is a better box defining strategy leading to smaller number of problems solved, hence smaller solution times, except the KP case with changing direction according to nadir. This good performance is mostly due to the increase in the number of boxes that are eliminated (E) with our elimination rule (see lines 26 and 28 of Algorithm 1).

We observe that the original algorithm (FDN) consistently performs good in terms of solution time over all test instances, being the fastest algorithm for KP and the second fastest for AP.

Table 2: Comparison of the Proposed Algorithms for class A of the set KP

| $N_{avg}$ | Algorithm | # IP | Run Time | # ($R(b,d)$) | Time per ($R(b,d)$) | Time per ($P_i(x^b)$) | $C$ | $C_2$ | $E$ |
|-----------|-----------|------|----------|--------------|---------------------|----------------------|-----|-----|-----|
| 975.4     | FDN       | 2541.20 | 838.06   | 1338.00      | 0.50                | 0.14                 | 233.20 | 7.40 | 595.00 |
|           | FDY       | 2762.80 | 947.70   | 1569.80      | 0.49                | 0.15                 | 223.20 | 7.60 | 362.80 |
|           | CDN       | 2398.60 | 894.72   | 1351.00      | 0.58                | 0.10                 | 72.60  | 2.40 | 592.00 |
|           | CDY       | 2512.60 | 1098.23  | 1520.40      | 0.62                | 0.14                 | 15.20  | 0.40 | 426.60 |
|           | NDN       | 2325.20 | 976.52   | 1347.60      | 0.65                | 0.10                 | 0.20   | 0.00 | 600.20 |
|           | NDY       | 2154.20 | 932.05   | 1176.80      | 0.67                | 0.14                 | 0.00   | 0.00 | 771.00 |

Table 3: Comparison of the Proposed Algorithms for class A of the set AP

| $N_{avg}$ | Algorithm | # IP | Run Time | # ($R(b,d)$) | Time per ($R(b,d)$) | Time per ($P_i(x^b)$) | $C$ | $C_2$ | $E$ |
|-----------|-----------|------|----------|--------------|---------------------|----------------------|-----|-----|-----|
| 708.4     | FDN       | 1636.20 | 2150.36  | 699.20       | 2.29                | 0.58                 | 246.60 | 20.00 | 674.60 |
|           | FDY       | 1978.00 | 2764.41  | 1056.60      | 2.09                | 0.60                 | 231.80 | 20.80 | 315.60 |
|           | CDN       | 1553.40 | 2187.07  | 712.00       | 2.43                | 0.54                 | 140.20 | 9.20  | 683.40 |
|           | CDY       | 2206.40 | 3924.58  | 1372.00      | 2.52                | 0.58                 | 132.40 | 8.40  | 25.00  |
|           | NDN       | 1431.00 | 2043.09  | 720.60       | 2.33                | 0.50                 | 0.00   | 0.00  | 653.20 |
|           | NDY       | 1862.20 | 2931.79  | 1151.80      | 2.22                | 0.53                 | 0.00   | 0.00  | 262.40 |

Based on these results, we conduct further preliminary experiments with FDN, CDN and NDN variants. Since finding the whole set of nondominated points might be computationally demanding for most biobjective integer programming problems, early termination performances of the algorithms are also worth considering. Therefore, we run these variants with predetermined time limits and report the quality of the set of nondominated points obtained. We calculate the coverage error measure, given by Definition 2.4.

Table 4 shows the performance results for the three algorithm variants when they are run with time limits for class A of KP and AP. The time limit is set as 300 and 700 seconds for KP and AP, respectively. This corresponds to approximately 30% of time required to find the whole set of nondominated points. The table shows the number of nondominated points found.
(\(\bar{N}\)), the coverage error (CE) and the scaled coverage error (SCE) values for each variant. It is seen that CDN significantly outperforms the other variants with respect to coverage. This result is expected as setting the direction as the diagonal vector of the box to be searched encourages the algorithm to find scattered solutions across the Pareto frontier and provides a highly representative set even at the early stages of the algorithm. In Figures 7-9, we provide the solution sets found when KP instances are solved with time limited versions of FDN, CDN and NDN, respectively.

Table 4: Coverage results with time limits for class A instances

| KP | Algorithm | Problem | \(\bar{N}\) | CE | SCE  | Algorithm | Problem | \(\bar{N}\) | CE | SCE  |
|----|-----------|---------|-------------|-----|-------|-----------|---------|-------------|-----|-------|
|    | FDN       | 1       | 478         | 408 | 0.1278| 1         | 230     | 727         | 0.2826|
|    |           | 2       | 528         | 324 | 0.0964| 2         | 227     | 724         | 0.2883|
|    |           | 3       | 438         | 451 | 0.1167| 3         | 231     | 663         | 0.2665|
|    |           | 4       | 559         | 369 | 0.1010| 4         | 227     | 711         | 0.2609|
|    |           | 5       | 452         | 333 | 0.1013| 5         | 220     | 771         | 0.2851|
|    | CDN       | 1       | 525         | 12  | 0.0038| 1         | 262     | 23          | 0.0089|
|    |           | 2       | 562         | 14  | 0.0042| 2         | 263     | 23          | 0.0085|
|    |           | 3       | 458         | 25  | 0.0065| 3         | 274     | 20          | 0.0081|
|    |           | 4       | 605         | 9   | 0.0025| 4         | 275     | 21          | 0.0088|
|    |           | 5       | 500         | 15  | 0.0046| 5         | 275     | 24          | 0.0089|
|    | NDN       | 1       | 362         | 552 | 0.1729| 1         | 229     | 757         | 0.2942|
|    |           | 2       | 418         | 514 | 0.1529| 2         | 224     | 784         | 0.2906|
|    |           | 3       | 311         | 628 | 0.1625| 3         | 230     | 705         | 0.2834|
|    |           | 4       | 432         | 571 | 0.1563| 4         | 239     | 757         | 0.2778|
|    |           | 5       | 304         | 540 | 0.1642| 5         | 231     | 794         | 0.2936|

In the first set of preliminary experiments, we eliminated the variants defining the box using \(y^b\), and concluded that FDN, CDN, NDN are worth further consideration, FDN being
the most computationally efficient one. In the second set of experiments with time limits we have observed that CDN is the top-performer. Based on these preliminary observations, we decided to perform the main experiments with FDN as it is computationally more promising, and CDN as it outperforms the other variants under time restriction.

Tables 5 and 6 show the results of our main experiments, in which we compare FDN and CDN over all instances of KP and AP. We report the average values for the number of nondominated solutions ($N_{avg}$), the number of models solved, the total solution times, the number of ($R(b,d)$) models solved and average time spent to solve ($R(b,d)$) and ($P_i(x^b)$) models as well as $C$, $C_2$ and $E$. The results verify the observations made at the preliminary experiments: although FDN solves more integer programming problems in total, it solves less of the more difficult ($R(b,d)$) models, hence it works faster than CDN. Moreover, when FDN is used, the number of cases where $y^b = n^b$ ($C$) is significantly larger than that of CDN. This is due to the nature of the direction vector used; moving along the search region in a fixed direction of $(1,1)^T$, the algorithm visits integer corners more often compared to a diagonal direction vector. This increases the cases where ($P_1(x^b)$) and ($P_2(x^b)$) are both solved within a box. Note that, in only a small portion of these cases two new nondominated points are found, implying that one ($P_1(x^b)$) is solved unnecessarily. However, since these models are much easier to solve compared to ($R(b,d)$), solving more of these does not significantly affect the computational performance of FDN.

Table 5: Results of the main experiments on KP

| Class: $N_{avg}$ | Algorithm | # IP | Run Time | # $R(b,d)$ | Time per $R(b,d)$ | Time per ($P_i(x^b)$) | $C$ | $C_2$ | $E$ |
|------------------|-----------|------|----------|------------|-------------------|-----------------------|-----|------|-----|
| A: 975.4         | FDN       | 2541.20 | 838.06   | 1338.00    | 0.50              | 0.14                  | 233.20 | 7.40 | 595.00 |
|                  | CDN       | 2398.60 | 894.72   | 1351.00    | 0.58              | 0.10                  | 72.60  | 2.40 | 592.00 |
| B: 1539.4        | FDN       | 3913.00 | 1546.16  | 1984.20    | 0.62              | 0.15                  | 409.80 | 22.40 | 1046.80 |
|                  | CDN       | 3704.00 | 2711.98  | 2027.80    | 1.04              | 0.33                  | 140.00 | 5.20 | 1037.60 |
| C: 2176.2        | FDN       | 5453.60 | 2539.96  | 2665.00    | 0.76              | 0.18                  | 657.20 | 46.80 | 1590.80 |
|                  | CDN       | 5152.20 | 3459.57  | 2744.00    | 1.12              | 0.16                  | 239.40 | 9.40 | 1586.60 |
| D: 2791.8        | FDN       | 6934.40 | 4605.52  | 3231.40    | 1.06              | 0.34                  | 995.20 | 86.00 | 2177.20 |
|                  | CDN       | 6503.20 | 5404.77  | 3345.80    | 1.43              | 0.19                  | 583.80 | 20.20 | 2194.40 |

Table 6: Results of the main experiments on AP

| Class: $N_{avg}$ | Algorithm | # IP | Run Time | # $R(b,d)$ | Time per $R(b,d)$ | Time per ($P_i(x^b)$) | $C$ | $C_2$ | $E$ |
|------------------|-----------|------|----------|------------|-------------------|-----------------------|-----|------|-----|
| A: 708.4         | FDN       | 1636.20 | 2150.36  | 609.20     | 2.29              | 0.58                  | 246.60 | 20.00 | 674.60 |
|                  | CDN       | 1553.40 | 2187.07  | 712.00     | 2.43              | 0.54                  | 140.20 | 9.20 | 683.40 |
| B: 1416.2        | FDN       | 3247.20 | 5554.08  | 1475.80    | 2.85              | 0.64                  | 379.20 | 26.00 | 1301.60 |
|                  | CDN       | 3096.20 | 5519.86  | 1506.20    | 3.02              | 0.60                  | 177.60 | 5.80 | 1311.60 |
| C: 823.6         | FDN       | 1895.00 | 5044.20  | 803.60     | 5.22              | 1.32                  | 288.80 | 23.00 | 794.60 |
|                  | CDN       | 1839.40 | 11212.29 | 815.80     | 12.04             | 1.33                  | 210.60 | 12.60 | 803.20 |
| D: 1827          | FDN       | 4140.20 | 16403.48 | 1808.20    | 6.95              | 1.64                  | 561.40 | 58.40 | 1726.00 |
|                  | CDN       | 3980.40 | 17451.84 | 1860.40    | 7.61              | 1.54                  | 304.00 | 13.00 | 1764.60 |

We also provide a comparison with the balanced box algorithm ([1]). Since the algorithms are coded and run on different platforms, we cannot compare the solution times with those of the balanced box algorithm. Note that the balanced box algorithm solves exactly $3N$ problems. Therefore, it will solve more models for all of the problem instances considered; indeed it solves 25.5%, 36.5% more problems than our best algorithm variant on average for KP and AP, respectively.

CDN works slower compared to FDN but our preliminary experiments show that it is
promising when used with time limits. We verified this observation by performing experiments for the whole KP and AP sets with time limit, the results of which are provided in Table 7.

| Problem | Time | FDN N CE SCE | CDN N CE SCE | Time | FDN N CE SCE | CDN N CE SCE |
|---------|------|---------------|---------------|------|---------------|---------------|
| 1       | 300  | 478 0.1278    | 525 0.0038    | 230  | 727 0.2826    | 262 0.0089    |
| 2       | 528  | 324 0.0064    | 562 0.0042    | 227  | 724 0.2683    | 263 0.0085    |
| 3       | 438  | 451 0.1167    | 458 0.0065    | 700  | 231 0.2665    | 274 0.0080    |
| 4       | 559  | 369 0.1010    | 605 0.0025    | 227  | 711 0.2609    | 275 0.0088    |
| 5       | 452  | 333 0.1013    | 500 0.0046    | 220  | 771 0.2851    | 275 0.0089    |
| 6       | 700  | 863 0.0904    | 847 0.0028    | 481  | 2566 0.3154   | 634 0.0045    |
| 7       | 748  | 528 0.0096    | 686 0.0037    | 478  | 2420 0.3150   | 631 0.0046    |
| 8       | 728  | 413 0.0860    | 763 0.0031    | 479  | 2543 0.3198   | 624 0.0048    |
| 9       | 821  | 444 0.0986    | 837 0.0040    | 476  | 2361 0.3044   | 604 0.0053    |
| 10      | 894  | 384 0.0836    | 815 0.0041    | 484  | 2507 0.3108   | 614 0.0040    |
| 11      | 1024 | 560 0.0938    | 809 0.0049    | 360  | 548 0.2207    | 361 0.0117    |
| 12      | 930  | 597 0.1011    | 712 0.0032    | 365  | 586 0.2224    | 124 0.0250    |
| 13      | 905  | 714 0.1080    | 754 0.0032    | 388  | 578 0.2284    | 251 0.0146    |
| 14      | 1156 | 637 0.1033    | 955 0.0036    | 382  | 613 0.2348    | 251 0.0134    |
| 15      | 975  | 464 0.0844    | 844 0.0029    | 406  | 530 0.2206    | 244 0.0137    |
| 16      | 1500 | 665 0.0920    | 1226 0.0022   | 611  | 3209 0.3255   | 755 0.0058    |
| 17      | 1266 | 650 0.0942    | 963 0.0045    | 603  | 3027 0.3102   | 777 0.0058    |
| 18      | 1370 | 756 0.0986    | 1066 0.0016   | 643  | 3039 0.3149   | 766 0.0051    |
| 19      | 1240 | 734 0.1044    | 899 0.0031    | 624  | 3058 0.3157   | 752 0.0056    |
| 20      | 1345 | 721 0.1042    | 1218 0.0017   | 605  | 3053 0.3161   | 770 0.0057    |

Table 7: Coverage results with time limits for the full set of problem instances

Overall, one can conclude that both variants are powerful in different aspects. When used to find the complete set of nondominated points, FDN works better since it runs faster. However, CDN is a very promising variant when run with a time limit since it quickly provides a highly representative subset of solutions.

5.1 An extension of CDN

When we examine the results of average time spend for an \((R(b,d))\) model, we observe that there is significant difference between FDN and CDN for class C of AP, see Table 6. In these instances, average time spent per \((R(b,d))\) model in CDN is more than twice of the time spent in FDN. To investigate this further, we check the solution times of each individual \((R(b,d))\) model solved in CDN for these instances. We see that the majority of the total time is occupied by only few models. To overcome this issue of extreme solution times, we modify CDN and solve each \((R(b,d))\) model under a time limit. If the model is aborted due to the time limit, we slightly modify the direction and solve the model with the new direction parameter. That is, we change line 4 of Algorithm 1 as follows:

We refer to this extension of CDN with time limited \((R(b,d))\) models as TL-CDN. We compare the performance of TL-CDN (where a time limit of 50 seconds is used for each \((R(b,d))\) model) with those of FDN and CDN in class C of AP. The results are presented in Table 8. When we compare the number of integer programming problems solved by the algorithms, we observe that CDN is the best algorithm and it is closely followed by TL-CDN, as expected. When we analyse the run times and average \((R(b,d))\) solution times of TL-
Replacement of line 4 of Algorithm 1

Let \( b(s^b, p^b, t^b) \in B \) and \( d = (p_1^b - s_1^b, t_2^b - s_2^b)^T \), attempt to solve \( (R(b,d)) \)
if \( (R(b,d)) \) could not be solved within the time limit then
\[ d_b^2 = d_b^2 - 1 \]
Solve \( (R(b,d)) \)

CDN and CDN, we observe that there is a significant improvement when TL-CDN is used, indicating that the extension is successful.

Table 8: Comparison of the FDN, CDN and TL-CDN for class C of the set AP

| Instance: \( N_{avg} \) | Algorithm | \# IP | Run Time | \# \( R(b,d) \) | Time per \( R(b,d) \) | Time per \( (P_i(x^b)) \) | \( C \) | \( C_2 \) | \( E \) |
|-------------------------|-----------|-------|----------|---------------|-------------------|---------------------------|-----|-----|-----|
| 11:813                  | FDN       | 1886  | 5789.17  | 797           | 5.38              | 1.37                      | 292 | 18  | 790 |
|                         | TL-CDN    | 1794  | 6888.21  | 807           | 6.96              | 1.28                      | 186 | 14  | 792 |
|                         | CDN       | 1790  | 7171.94  | 803           | 7.32              | 1.31                      | 186 | 14  | 792 |
| 12:827                  | FDN       | 1872  | 5704.50  | 808           | 5.30              | 1.33                      | 261 | 26  | 791 |
|                         | TL-CDN    | 1867  | 7406.11  | 844           | 7.24              | 1.26                      | 203 | 9   | 805 |
|                         | CDN       | 1848  | 12428.33 | 827           | 13.36             | 1.34                      | 202 | 10  | 804 |
| 13:823                  | FDN       | 1915  | 5655.16  | 805           | 5.21              | 1.31                      | 305 | 20  | 798 |
|                         | TL-CDN    | 1869  | 6699.22  | 825           | 6.57              | 1.22                      | 226 | 7   | 812 |
|                         | CDN       | 1860  | 8182.69  | 817           | 8.41              | 1.25                      | 225 | 7   | 812 |
| 14:8411                 | FDN       | 1923  | 5715.18  | 817           | 5.17              | 1.34                      | 290 | 27  | 808 |
|                         | TL-CDN    | 1892  | 6875.17  | 840           | 6.58              | 1.27                      | 225 | 16  | 815 |
|                         | CDN       | 1883  | 14560.13 | 832           | 15.70             | 1.41                      | 224 | 16  | 815 |
| 15:814                  | FDN       | 1879  | 5356.99  | 791           | 5.03              | 1.26                      | 296 | 24  | 786 |
|                         | TL-CDN    | 1832  | 7023.40  | 814           | 7.09              | 1.23                      | 218 | 16  | 793 |
|                         | CDN       | 1816  | 13718.39 | 800           | 15.41             | 1.36                      | 216 | 16  | 793 |

We also run TL-CDN with predetermined time limits for class C of AP and report the quality of the solution set (using coverage error) in Table 9, by comparing it with FDN and CDN. It is seen that TL-CDN outperforms CDN in terms of coverage error in most instances. The results show that this modification (TL-CDN) is successful in significantly reducing run time without sacrificing from performance in representativeness.

Table 9: Coverage results with time limits for class C of AP instances

| Problem | \( \bar{N} \) | CE | SCE | \( \bar{N} \) | CE | SCE | \( \bar{N} \) | CE | SCE |
|---------|-------------|----|-----|-------------|----|-----|-------------|----|-----|
| 1       | 360         | 0.2207 | 0.2207 | 361         | 0.2207 | 0.2207 | 384         | 0.2207 | 0.2207 |
| 2       | 365         | 0.2224 | 0.2224 | 124         | 0.2224 | 0.2224 | 348         | 0.2224 | 0.2224 |
| 3       | 388         | 0.2284 | 0.2284 | 251         | 0.2284 | 0.2284 | 372         | 0.2284 | 0.2284 |
| 4       | 382         | 0.2348 | 0.2348 | 251         | 0.2348 | 0.2348 | 392         | 0.2348 | 0.2348 |
| 5       | 406         | 0.2206 | 0.2206 | 244         | 0.2206 | 0.2206 | 343         | 0.2206 | 0.2206 |
6 Conclusion

We propose an exact solution approach for biobjective integer programming problems based on solving Pascoletti-Serafini scalarizations to search for nondominated points within boxes in the objective space. We generate different variants of the algorithm based on how the boxes are defined and how the direction vector in the scalarization problem is set.

We compare the performances of the algorithm variants both with and without time limits and determine the ones that outperform the others. Our results indicate that the variants using nondominated points to define the boxes are better. Moreover, although the variant using a fixed direction vector of $(1, 1)^T$ leads to more (mixed) integer programming problems solved, it requires less computational time since it solves less of the more difficult scalarization models. We, however, observe that the variant setting direction with respect to the diagonal of the box to be searched is still promising since it returns a highly representative subset (measured using coverage error) of the set of nondominated points when it is run with a time limit. We suggest an extension to this variant, which has lower solution times and better coverage error results.

We prove that the algorithm terminates and show through computational experiments that the best variant solves less problems than a recent algorithm, the balanced box method.

References

[1] Natashia Boland, Hadi Charkhgard, and Martin Savelsbergh. A criterion space search algorithm for biobjective integer programming: The balanced box method. INFORMS Journal on Computing, 27(4):735–754, 2015.

[2] V. Joseph Bowman. On the relationship of the Tchebycheff norm and the efficient frontier of multiple-criteria objectives. In Hervé Thiriez and Stanley Zionts, editors, Multiple Criteria Decision Making, pages 76–86, Berlin, Heidelberg, 1976. Springer Berlin Heidelberg.

[3] Gkhan Ceyhan, Murat Kksalan, and Banu Lokman. Finding a representative nondominated set for multi-objective mixed integer programs. European Journal of Operational Research, 272(1):61 – 77, 2019.

[4] L.G. Chalmet, L. Lemonidis, and D.J. Elzinga. An algorithm for the bi-criterion integer programming problem. European Journal of Operational Research, 25(2):292 – 300, 1986.

[5] V. Chankong and Y.Y. Haimes. Multiobjective Decision Making Theory and Methodology. Elsevier Science, New York, 1983.

[6] Rui Dai and Hadi Charkhgard. A two-stage approach for bi-objective integer linear programming. Operations Research Letters, 46(1):81 – 87, 2018.

[7] Matthias Ehrgott. Multicriteria Optimization (2. ed.). Springer, 2005.

[8] Y Haimes, L Lasdon, and D Wismer. On a bicriterion formulation of the problems of integrated system identification and system optimization. IEEE Transactions on Systems, Man, and Cybernetics, SMC-1(3):296–297, 1971.
[9] Juhani Koski. *Multicriteria Truss Optimization*, pages 263–307. Springer US, Boston, MA, 1988.

[10] A. Pascoletti and P. Serafini. Scalarizing vector optimization problems. *Journal of Optimization Theory and Applications*, 42(4):499–524, Apr 1984.

[11] Anthony Przybylski, Xavier Gandibleux, and Matthias Ehrgott. Two phase algorithms for the bi-objective assignment problem. *European Journal of Operational Research*, 185(2):509 – 533, 2008.

[12] Ted Ralphs, Matthew Saltzman, and Margaret Wiecek. An improved algorithm for solving biobjective integer programs. *Annals OR*, 147:43–70, 10 2006.

[13] Serpil Sayın. Measuring the quality of discrete representations of efficient sets in multiple objective mathematical programming. *Mathematical Programming*, 87(3):543–560, May 2000.

[14] Serpil Sayın and Panos Kouvelis. The multiobjective discrete optimization problem: A weighted min-max two-stage optimization approach and a bicriteria algorithm. *Management Science*, 51(10):1572 – 1581, 2005.

[15] Ralph E. Steuer and Eng-Ung Choo. An interactive weighted Tchebycheff procedure for multiple objective programming. *Mathematical Programming*, 26:326–344, 10 1983.

[16] E.L. Ulungu and J. Teghem. The two phases method: An efficient procedure to solve biobjective combinatorial optimization problems. *Foundations of Computing and Decision Sciences*, 20(2):149–165, 1995.

[17] L. Zadeh. Optimality and non-scalar-valued performance criteria. *IEEE Transactions on Automatic Control*, 8(1):59–60, 1963.