Markus Leitner[1] Ivana Ljubić[2] Markus Sinnl[1]
Axel Werner[3]

Two algorithms for solving 3-objective k-ArchConFL and IPs in general

[1] University of Vienna, e-mail: markus.{leitner,sinnl}@univie.ac.at
[2] ESSEC Business School Cergy-Pontoise, e-mail: ivana.ljubic@essec.edu
[3] Zuse Institute Berlin, e-mail: werner@zib.de, supported by DFG within project GR 883/18-1

ZIB Report 15-48 (September 2015)

Herausgegeben vom
Konrad-Zuse-Zentrum für Informationstechnik Berlin
Takustraße 7
D-14195 Berlin-Dahlem

Telefon: 030-84185-0
Telefax: 030-84185-125

e-mail: `bibliothek@zib.de`
URL: `http://www.zib.de`

ZIB-Report (Print) ISSN 1438-0064
ZIB-Report (Internet) ISSN 2192-7782

Two algorithms for solving 3-objective k-ArchConFL and IPs in general

Markus Leitner[*] Ivana Ljubić[†] Markus Sinnl[‡]
Axel Werner[§]

September 29, 2015

1 Introduction

Planning access networks is a complex process that involves strategic business decisions, technical considerations, and more. Mathematical models usually focus only on parts of the planning process, as it is virtually impossible to integrate all choices into a unified model; see [5] for more details and literature. In this report, we discuss the *k-Architecture Connected Facility Location Problem* [7, 8], which involves the strategic decisions of which customers to provide with which technology (or architecture), as well as which routes and locations to use for setting up new hardware in the deployment area. For technologies, the choice is usually between (new) fiber connections, (existing) copper lines and possibly wireless links, but other classifications – such as different quality-of-service levels for copper connections – are also conceivable. Apart from the distinction between these technologies, no technical details, such as capacities of devices, cable sizes, etc., are considered.
Access network planning problems are naturally multiobjective problems. Apart from minimizing the cost of the network, an operator would also strive to provide connections to as many (potential) customers as possible to maximize their revenue. Hence it makes sense to treat the planning problem as a 3-objective optimization problem, where the total cost is minimized and, at the same time, the total coverage within the deployment area as well as the coverage with the "preferred" technology (which would, in a realistic setting, most probably be fiber connections) are maximized. This means that for solving such problems, 3-objective optimization algorithms have to be employed. In Section 3,

[*]University of Vienna, e-mail: markus.leitner@univie.ac.at
[†]ESSEC Business School Cergy-Pontoise, e-mail: ivana.ljubic@essec.edu
[‡]University of Vienna, e-mail: markus.sinnl@univie.ac.at
[§]Zuse Institute Berlin, e-mail: werner@zib.de

we present two such algorithms, which both share some similarities with known methods, see [2, 3], and [6]. Concluding, we report on some results of computations using small test instances.

Terminology and Notation. Throughout the paper, the architectures are simply numbered $1, \dots, k$ and architecture 1 is considered to be the "preferred" technology.
For terminology in multiobjective optimization in general, we refer to the literature, for instance, Ehrgott's book [4]. In addition we make use of the following two pieces of notation (mainly in $\mathbb{R}^3$ instead of $\mathbb{R}^n$, though):

- For two points $x, y \in \mathbb{R}^n$, where $x = (x_1, \dots, x_n)$ and $y = (y_1, \dots, y_n)$,

$$x \leqslant y \quad :\Longleftrightarrow \quad x_i \leqslant y_i \quad \forall i = 1, \dots, n.$$

- With $[x, y]$ (x and y as above) we denote the set

$$[x, y] := [x_1, y_1] \times \cdots \times [x_n, y_n],$$

the (n-dimensional) box between x and y; note that $[x, y] = \varnothing$ if $x \nleqslant y$.

2 3-objective k-ArchConFL

In the following, we present a 3-objective binary integer program (IP) that models the access network planning problem with k architectures. The model aims to

(i) minimize the total setup cost of the network,

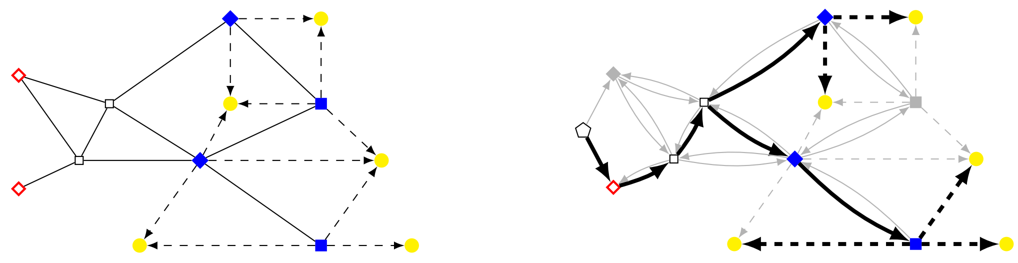
(ii) maximize the demand that is satisfied by connection with the preferable architecture,

(iii) maximize the demand satisfied in total (by any architecture).

The model allows for a relatively straightforward way to compute the ideal and nadir point, at least in the practically relevant cases. This strategy as well as some general observations are described in subsequent sections.

2.1 Input data

Notation of parameters for an instance to the 3-objective k-ArchConFL problem follows the description in [7]. An instance consists of an undirected *core graph* representing the possible routes and installation points of fiber connections and directed *assignment arcs* representing possible connections of customers to facilities; see Figure 1(a). Each edge in the undirected core graph is replaced by a forward and a backward arc, so that, in essence, we have a directed graph $G = (V, A)$ where the node set is the disjoint union $V = Q \cup C \cup F \cup S$ of


(a) Exemplary Instance: Two potential COs (red) and five customers (yellow) that can be connected to one of four facilities (blue), two for each of the two architectures (diamond and rectangle shaped, respectively)

(b) Exemplary Solution: Two customers are connected using architecture 1, forcing one facility to be opened, and three customers connected with architecture 2, also using one facility; one CO is opened and a facility for architecture 1 is used as a Steiner node

Figure 1: Exemplary instance and solution to the 3-objective k-ArchConFL problem ($k = 2$)

(i) potential central offices (COs) Q with opening costs $c_q \geqslant 0, \forall q \in Q$,

(ii) customer nodes C with demands $d_c \in \mathbb{N}, \forall c \in C$,

(iii) potential facility locations $F = \bigcup_{i=1}^{k} F^i$ with opening costs $c_i^l \geqslant 0, \forall i \in F^l$, $l = 1, \ldots, k$, and

(iv) potential Steiner nodes S.

Potential facilities in F^l represent locations where equipment can be installed to connect customers using architecture l; note that F^i and F^j need not be disjoint for $i \neq j$. The arc set A consists of

(i) the core arcs $A_c = \{(i,j) \in A \mid i,j \notin C\}$, with trenching costs $c_a \geqslant 0, \forall a \in A_c$, and

(ii) assignment arcs $A^l = \{(i,j) \in A \mid i \in F^l, j \in C\}$, for each architecture $l = 1, \ldots, k$, with costs $c_{ij}^l \geqslant 0$ for connecting customer j to facility i using architecture l.

We set $D := \sum_{j \in C} d_j$ to be the total demand of all customers. We also extend the graph G with an artificial root node $r \notin V$, connected via artificial arcs $A_r = \{(r,q) \mid q \in Q\}$ to all central offices, and for each artificial arc (r,q), $q \in Q$, we set their cost $c_{rq} := c_q$. For abbreviation we use $A_{rc} := A_r \cup A_c$.
Figure 1 shows a small illustrating example of an instance and a feasible solution.

2.2 Model

The following 3-objective binary IP in minimization form is basically the model from [7] for k (instead of only 2) architectures, where customer coverage is taken care of by suitable objectives (instead of given coverage rates).

$$
\begin{aligned}
\min \quad & z_1 := D - \sum_{j\in C} d_j z_j^1 \\
& z_k := D - \sum_{l=1}^{k}\sum_{j\in C} d_j z_j^l \\
& z_0 := \sum_{(i,j)\in A_{rc}} c_{ij}x_{ij} + \sum_{l=1}^{k}\sum_{(i,j)\in A^l} c_{ij}^l x_{ij}^l + \sum_{l=1}^{k}\sum_{i\in F^l} c_i^l y_i^l
\end{aligned}
$$

$$
\begin{aligned}
\text{s.t.} \quad & \sum_{l=1}^{k} z_j^l \leqslant 1 && \forall j\in C && (1)\\
& \sum_{i\in F_j^l} x_{ij}^l = z_j^l && \forall j \in C,\ l=1,\dots,k && (2)\\
& x_{ij}^l \leqslant y_i^l && \forall j\in C,\ i\in F_j^l,\ l=1,\dots,k && (3)\\
& x(\delta^-(W)) \geqslant y_i^l && \forall W\subseteq V\backslash C,\ i\in F^l\cap W,\ l=1,\dots,k && (4)\\
& x_a, y_i^l, z_j^l \in \{0,1\} && \forall a\in A_c,\ i\in F^l,\ j\in C,\ l=1,\dots,k && (5)\\
& x_{ij}^l \in\{0,1\} && \forall i\in F_j^l,\ j\in C,\ l=1,\dots,k && (6)
\end{aligned}
$$

(1) – (6) describe a Connected Facility Location model (see [7, Sec. 2] for more details) with some additional bookkeeping of the type of architecture used to connect the customers. This information is exploited in the first two objectives: z_1 is the total demand *not* covered by architecture 1, and z_k the total demand *not* covered by *any* architecture at all. Minimizing z_1 and z_k is obviosuly equivalent to

$$
\max \sum_{j\in C} d_j z_j^1 \qquad \text{and} \qquad \max \sum_{l=1}^{k}\sum_{j\in C} d_j z_j^l,
$$

respectively; hence these objectives drive solutions to connect many customers in general, and many with the best technology 1 in particular. The third objective z_0 represents the total cost of a solution, which should be minimized.

2.3 Computation of ideal and nadir point

For each feasible solution to the 3-objective k-ArchConFL problem, we obviously have $z_0 \geqslant 0$, due to non-negativity of cost coefficients, and $0 \leqslant z_k \leqslant z_1 \leqslant D$. Hence all nondominated points are located within the area given by

$$
\{(z_1, z_k, z_0) \mid (z_1, z_k) \in Z,\ z_0 \geqslant 0\} = Z \times \mathbb{R}^+
$$

in the objective space, where $Z = \text{conv}\{(0,0),(D,0),(D,D)\}$ (see Figure 2). This implies the obvious bound $(0,0,0)$ for the ideal point, as well as a bound $(D, D, \tilde{c})$ for the nadir

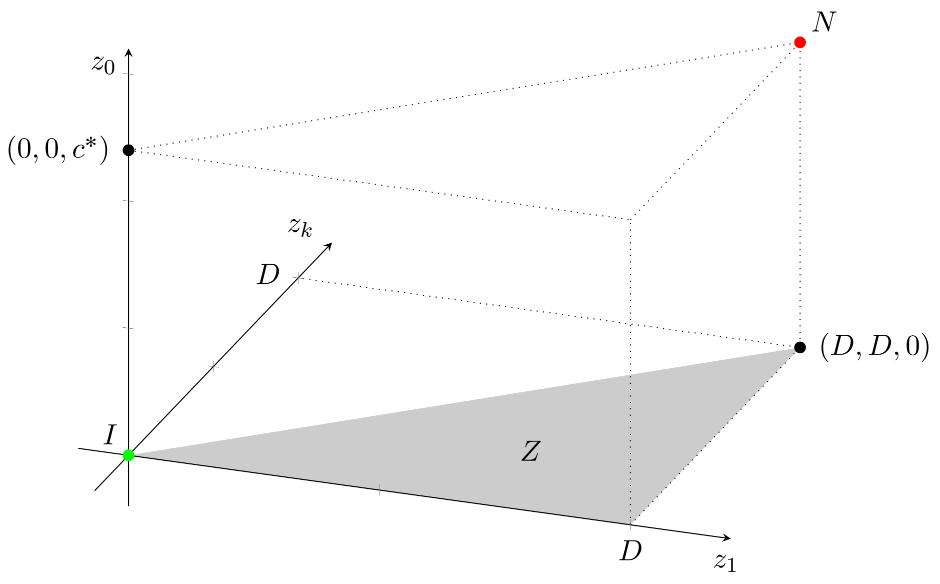


Figure 2: Ideal and nadir point in the case that every customer can be reached by architecture 1

point from any upper bound $\tilde{c}$ on the cost of a feasible solution, such as the trivial upper bound obtained by taking the sum of all involved cost values.
The exact ideal and nadir point, I and N, respectively, are – under mild assumptions – relatively easy to obtain. We assume that all costs for assignment arcs are strictly positive. Then $(D, D, 0)$ is a non-dominated point: It is the objective vector of the 0-solution, which is feasible for (1) – (6); furthermore, there is no other feasible solution with cost 0 that has $z_1 < D$ or $z_k < D$, since as soon as customers are connected, an assignment arc with non-zero cost value has to be opened.

Ubiquity of architecture 1. If every customer is reachable via assignment arcs of architecture 1 then there is a solution with $z_1 = z_k = 0$. Let c^* be the optimal cost of all such solutions, which can be obtained by solving the single-objective IP

$$\begin{aligned} \min \ & z_0 \\ & z_j^1 = 1 \qquad \forall j \in C \\ & (1) - (6) \end{aligned}$$

Then, obviously, $(0, 0, c^*)$ is a non-dominated point. Hence (cf. Figure 2)

$$I = (0, 0, 0)\ , \qquad N = (D, D, c^*).$$

General ubiquity. The above argument can easily be generalized to the case where every customer can be reached by at least *some* architecture, which does not necessarily

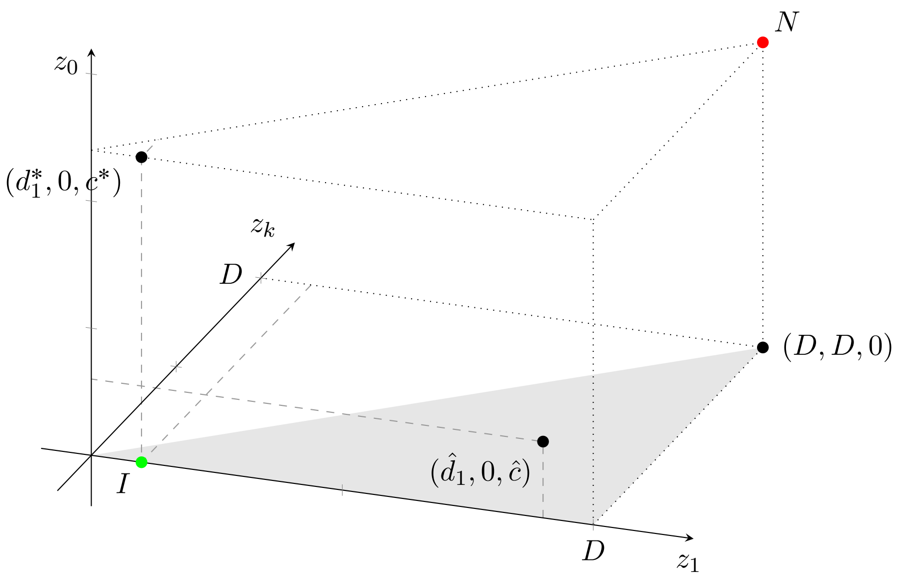


Figure 3: General case – every customer can be reached by some architecture

have to be architecture 1. Let $d_1^* := D - \sum_{j \in C_1} d_j$ with

$$C_1 := \{j \in C \mid j \text{ can be reached by architecture } 1\}$$

and c^* the optimal value of the following single-objective IP:

$$\begin{aligned} \min \ & z_0 \\ & z_j^1 = 1 && \forall j \in C_1 \\ & \sum_{l=1}^{k} z_j^l = 1 && \forall j \in C \\ & (2) - (6) \end{aligned}$$

Then $(d_1^*, 0, c^*)$ is a non-dominated point and there is no feasible solution with $z_1 < d_1^*$ (see Figure 3). Hence

$$I = (d_1^*, 0, 0) , \qquad N = (D, D, c^*).$$

Another non-dominated point $(\hat{d}_1, 0, \hat{c})$ can be computed relatively easy by forcing equality in (1), as above, and then sequentially solving the single-objective problems $\min z_0$ (giving $\hat{c}$) and $\min z_1$ with added constraint $z_0 \leqslant \hat{c}$.

Unreachable customers. If general ubiquity does not hold, then there are customers that cannot be connected at all. Without loss of generality, these customers can be removed in a preprocessing step. Hence we can always assume general ubiquity at the least.

Zero assignment cost values. If our initial assumption – all assignment costs are strictly positive – does not hold, there are assignment arcs with cost 0 and the point $(D, D, 0)$ is no longer nondominated in general. In fact, the z_1- and z_k-coordinates of the nadir point might be determined by two different nondominated points with $z_0 = 0$; small examples for this case can easily be constructed. Nevertheless, (D, D, c^*) is still an upper bound for the nadir point.

3 3-objective solution algorithms

We now describe two algorithms to solve the 3-objective k-ArchConFL problem. Both methods do not explicitly make use of the problem structure, hence they are applicable to any problem that can be formulated using a 3-objective IP model satisfying certain basic properties. Accordingly, we state the algorithms in a general way, without referring to 3-objective k-ArchConFL. As such, the three objectives are denoted in this section as f_1, f_2, and f_3, which, for the 3-objective k-ArchConFL, correspond to z_1, z_k, and z_0, respectively, as used in Section 2. Similarly, we frequently use x to denote points (and their components) in the 3-dimensional objective space, which is not to confuse with the x_a and x_{ij}^l variables in Section 2.

3.1 General remarks

In the following, we make a few basic assumptions:

- The IP considered is assumed to have only integer coefficients, which allows for a less complicated description of the algorithms. This can, for rational IPs, in general be enforced by multiplying the coefficients with a suitable integer value, but might lead to numerical problems which we ignore here, however.

- We assume that there is a subroutine available to compute the ideal and nadir point. Both methods still work if only lower and upper bounds, respectively, can be computed beforehand. For the 3-objective k-ArchConFL problem, ideal and nadir point can be obtained easily, as seen in Section 2.3.

- We further assume there is a solver available which can be used as a black box to solve occurring 3-objective sub-IPs. More precisely, the algorithms make frequent use of the two subroutines

  lexminsolve($\star$) for a given IP $\star$, which solves three consecutive single-objective IPs, first $\star$ minimizing f_3 to obtain a bound x_3, then $\star$ with added constraint $f_3 \leqslant x_3$ minimizing f_1 to obtain a bound x_1, and finally $\star$ with added constraints $f_3 \leqslant x_3$, $f_1 \leqslant x_1$ minimizing f_2 to obtain a nondominated point (x_1, x_2, x_3);

  weightedsumsolve($\star$) for a given IP $\star$, which solves the single-objective IP $\star$ minimizing a weighted sum $w_1 f_1 + w_2 f_2 + w_3 f_3$ with $w_1, w_2, w_3 \geqslant 0$; the weights are not given explicitly, since the methods work correctly with every possible such choice.

The performance of the algorithms depends not only on the performance of the used IP solver, but also on the actual implementation of subroutines such as *pop* or list insertion, the precise weights used in *weightedsumsolve*, and the meaning and computability of "best" in Algorithms 2 and 5.

3.2 Epsilon-constraint method

The first algorithm is a variant of the well-known epsilon-constraint method [1]. There exist a number of adaptations of the biobjective epsilon-constraint method for multiobjective problems; see, for instance, [6], where a multi-dimensional grid structure is used to keep track of those regions of the objective space already ruled out, as well as [3] for more references. In our variant, we generalize the sequential 2-objective approach to 3 objectives (which can straightforwardly be generalized to any number of objectives), thereby making use of further parameters, which we call validity ranges, to speed up the process.

In the following we explain how the algorithm proceeds, see Algorithm 1. After initialization of the ideal and nadir point, I and N, (alternatively, appropriate bounds for I and N), two initially empty sets are introduced: the nondominated set S, which is the final result and returned at the end, as well as a list C of candidate points that are computed during the algorithm and may be reused to avoid finding the same nondominated point multiple times. In the main part, a bound ε_1 (initialized with the first coordinate value of N) is iteratively decreased in an outer loop (lines 6–31). In every iteration, a second bound ε_2 (initialized with the second coordinate value of N) is decreased in the same manner (lines 8–29). Both of these bounds are used in every iteration of the inner loop to obtain a nondominated point x^* by either solving the given IP using the *lexminsolve* subroutine (line 11 or 16) or finding a "best" solution out of the already computed ones that is feasible to the current ε bounds (line 9, see below). If no nondominated point was found (lines 18–19), the method terminates by setting both bounds below their abort value. Otherwise the maintained sets are updated if applicable (lines 21–26) and the inner ε bound is decreased according to the second objective value of the found solution (line 27).

For finding and handling a solution in a given iteration, there are two possibilities: either solving an IP augmented with epsilon-constraints or using a nondominated point from a solution obtained earlier. Let x^* be a nondominated point that was found by the subroutine

$$\mathit{lexminsolve}(\bigstar, f_1 \leqslant \alpha_1, f_2 \leqslant \alpha_2),$$

i. e., by solving the IP obtained by augmenting $\bigstar$ with the constraints $f_i \leqslant \alpha_i$, $i = 1, 2$ for some bounds α_1 and α_2. Then we call (α_1, α_2) the *validity range of* x^*. Then a validity range of (α_1, α_2) of some candidate point $\bar{x}$ implies that any subsequent call to $\mathit{lexminsolve}(\bigstar, f_1 \leqslant \varepsilon_1, f_2 \leqslant \varepsilon_2)$, where $\alpha_1 \geqslant \varepsilon_1$ and $\alpha_2 \geqslant \varepsilon_2$ (i. e., the validity range of $\bar{x}$ exceeds the current ε bounds), but at the same time $\bar{x}_1 \leqslant \varepsilon_1$ and $\bar{x}_2 \leqslant \varepsilon_2$ holds

Algorithm 1 Epsilon-constraint method for 3 objectives

```
Input: IP model (IP) with 3 objectives $f_1$, $f_2$, $f_3$
Output: nondominated set $S \subset \mathbb{Z}^3$
 1: $I = (x_1^I, x_2^I, x_3^I) \leftarrow$ compute ideal point
 2: $N = (x_1^N, x_2^N, x_3^N) \leftarrow$ compute nadir point
 3: $S \leftarrow \varnothing$                                   // nondominated set
 4: $C \leftarrow \varnothing$          // list of candidate points with validity ranges
 5: $\varepsilon_1 \leftarrow x_1^N$
 6: while $\varepsilon_1 \geqslant x_1^I$ do
 7:     $\varepsilon_2 \leftarrow x_2^N$
 8:     while $\varepsilon_2 \geqslant x_2^I$ do
 9:         $(\bar{x}, \alpha_1, \alpha_2) \leftarrow \mathit{bestcandidate}(C, \varepsilon_1, \varepsilon_2)$
10:         if $\bar{x} =$ null then
11:             $x^* \leftarrow \mathit{lexminsolve}((\mathrm{IP}), f_1 \leqslant \varepsilon_1, f_2 \leqslant \varepsilon_2)$
12:         else if $\alpha_1 \geqslant \varepsilon_1$ and $\alpha_2 \geqslant \varepsilon_2$ then
13:             $x^* \leftarrow \bar{x}$
14:         else
15:             set $\bar{x}$ as start solution
16:             $x^* \leftarrow \mathit{lexminsolve}((\mathrm{IP}), f_1 \leqslant \varepsilon_1, f_2 \leqslant \varepsilon_2)$
17:         end if
18:         if $x^* =$ null then
19:             $\varepsilon_1 \leftarrow x_1^I - 1$, $\varepsilon_2 \leftarrow x_2^I - 1$     // terminate: no solution left
20:         else
21:             if $x^* = \bar{x}$ then  // same point with better (or same) validity range
22:                 $C \leftarrow C \backslash \{(\bar{x}, \alpha_1, \alpha_2)\} \cup \{(x^*, \varepsilon_1, \varepsilon_2)\}$
23:             else
24:                 $S \leftarrow S \cup \{x^*\}$                      // new solution
25:                 $C \leftarrow C \cup \{(x^*, \varepsilon_1, \varepsilon_2)\}$
26:             end if
27:             $\varepsilon_2 \leftarrow x_2^* - 1$
28:         end if
29:     end while
30:     $\varepsilon_1 \leftarrow \max\{x_1 \mid (x_1, x_2, x_3) \in S, x_1 < \varepsilon_1\} - 1$
31: end while
32: return $S$
```

(i. e., $\bar{x}$ is feasible for the IP treated in the call), will only yield $\bar{x}$ as a nondominated point again. In other words, under these circumstances, a new call to *lexminsolve*, and therefore solving the IP again, can be avoided and the candidate point $\bar{x}$ itself be taken (line 13). If no candidate with a sufficiently big validity range can be found, then a feasible candidate can at least be used to set a start solution for solving the IP (line 15). The selection of the candidate is described in Algorithm 2, where the meaning of "best" is not explicitly defined – various strategies are conceivable.

Algorithm 2 *bestcandidate*

```
Input: set $C$ containing objective points with validity ranges, bounds $\varepsilon_1$, $\varepsilon_2$
Output: objective point with validity range (possibly null)
1: if $C = \emptyset$ or $\nexists(x, \alpha_1, \alpha_2) \in C : x_1 \leqslant \varepsilon_1, x_2 \leqslant \varepsilon_2$ then
2:     return null
3: else if $\exists(x, \alpha_1, \alpha_2) \in C : x_1 \leqslant \varepsilon_1, x_2 \leqslant \varepsilon_2, \alpha_1 \geqslant \varepsilon_1, \alpha_2 \geqslant \varepsilon_2$ then
4:     return $(x, \alpha_1, \alpha_2)$
5: else
6:     return any $(x, \alpha_1, \alpha_2) \in C$ with $x_1 \leqslant \varepsilon_1, x_2 \leqslant \varepsilon_2$ // preferably the "best" such
       point
7: end if
```

Any found solution yields a nondominated point, hence new points are always added to the nondominated set and the candidate list (lines 24 and 25). If a point has the same objective values as the retrieved candidate, it does not have to be added to S again, but can replace the old candidate point and its (possibly worse) validity range in the candidate list C (line 22).

Remarks. The *lexminsolve* subroutine can be replaced by *weightedsumsolve* without compromising the correctness of the algorithm. Our computational experience with the 3-objective k-ArchConFL suggests that this has no clear impact on the performance of the algorithm. This might, however, be due to the special structure of the problem (values of the f_3/cost objective are larger than of the other objectives by orders of magnitude) and may be different for other types of IPs.
For the 3-objective k-ArchConFL model, the ε_2 bound can be initialized in each inner loop with the value ε_1, due to the fact that all feasible solutions satisfy $f_2 \leqslant f_1$, or, in the notation of Section 2, $z_k \leqslant z_1$.

3.3 Box-splitting algorithm

The second algorithm, which we present in this section, works by subsequently subdividing the objective space into smaller boxes that might contain still undiscovered nondominated points. In essence, it computes the points in a similar fashion as the Full m-split algorithm of Dächert & Klamroth, see [2, 3], but without making use of the neighor relation exhibited there. Instead, possibly redundant parts of the search region are skipped by postponing the solution of the associated boxes and reducing them

by slicing or removing them altogether, before they are eventually examined (by being upgraded to high priority).
The main procedure is given in Algorithm 3. It maintains two lists of boxes that describe

Algorithm 3 Box-splitting algorithm for 3 objectives

```
Input: IP model (IP) with 3 objectives $f_1$, $f_2$, $f_3$
Output: non-dominated frontier $S \subset \mathbb{Z}^3$
 1: $I = (x_1^I, x_2^I, x_3^I) \leftarrow$ compute ideal point
 2: $N = (x_1^N, x_2^N, x_3^N) \leftarrow$ compute nadir point
 3: $S \leftarrow \varnothing$                                  // non-dominated frontier
 4: $H \leftarrow \{[I, N]\}$                               // list of high-priority boxes
 5: $L \leftarrow \varnothing$                                  // list of low-priority boxes
 6: while $H \cup L \neq \varnothing$ do
 7:     if $H \neq \varnothing$ then
 8:         $B = [a, b] \leftarrow pop(H)$
 9:         $x^* \leftarrow weightedsumsolve((\text{IP}), a_i \leqslant f_i \leqslant b_i, i = 1, \ldots, 3)$
10:         if $x^* \neq$ null then
11:             $S \leftarrow S \cup \{x^*\}$
12:             $splitbox(B, x^*, H, L)$             // adds new boxes to $H$ and $L$
13:         end if
14:     else                                       // no high-priority box left
15:         $B = [a, b] \leftarrow pop(L)$
16:         $\bar{x} \leftarrow bestdominating(B, S)$
17:         if $\bar{x} =$ null then                     // no part of $B$ is dominated
18:             $H \leftarrow H \cup \{B\}$
19:         else if $\bar{x}_i \leqslant a_i \; \forall i = 1, \ldots, 3$ then
20:             do nothing                   // $B$ is completely dominated by $\bar{x}$
21:         else
22:             $slicebox(B, \bar{x}, L)$
23:         end if
24:     end if
25: end while
26: return $S$
```

a partition of the part of the objective space, in which new nondominated points might still be found. The first list, H, contains the *high-priority boxes*, whereas the second, L, contains the *low-priority boxes*. After initialization, H contains the complete box given by the ideal and nadir point (line 4), whereas L is empty (line 5). In the following main loop, the high-priority boxes are examined (lines 7–13), where new boxes are created, of high as well as low priority, until there are no more of them available; only then the low-priority boxes are treated (lines 14–23).
For each high-priority box B, retrieved from the list in line 8, the given IP is solved within B using the *weightedsumsolve* subroutine (line 9). If this yields a nondominated point, it is added to the nondominated set (line 11) and subsequently the box is splitted

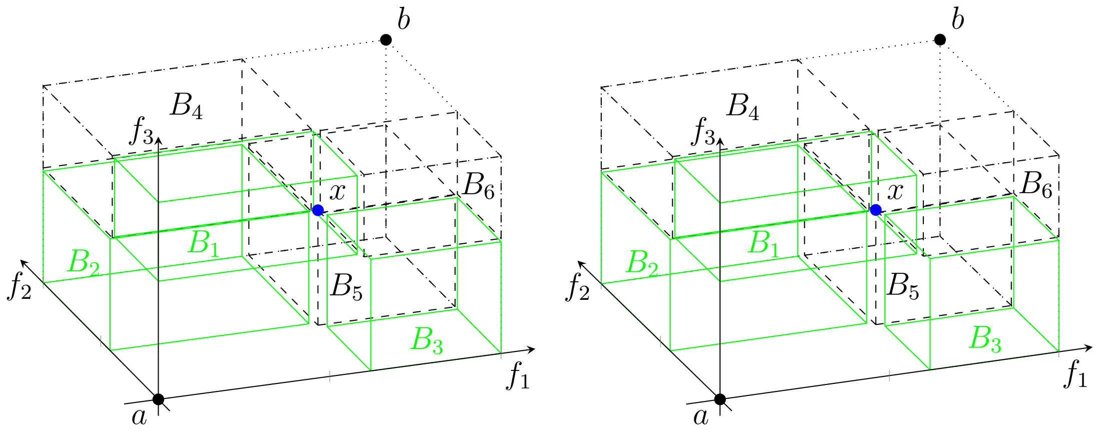


Figure 4: High-priority boxes (green) and low-priority boxes (dashed) determined by a nondominated point x (3d picture)

into smaller boxes. This is done by Algorithm 4. The point found, denoted by x in

Algorithm 4 *splitbox*

Input: box $B = [a, b]$, point $x \in B$, lists H (high-priority boxes) and L (low-priority boxes)

Output: — (adds up to 6 new boxes to the lists H, L)

```
1: $B_1 \leftarrow [(a_1, a_2, x_3 + 1), (x_1 - 1, x_2 - 1, b_3)]$        // might be empty
2: $B_2 \leftarrow [(a_1, x_2 + 1, a_3), (x_1 - 1, b_2, x_3 - 1)]$        // might be empty
3: $B_3 \leftarrow [(x_1 + 1, a_2, a_3), (b_1, x_2 - 1, x_3 - 1)]$        // might be empty
4: $H \leftarrow H \cup \{B_i \mid B_i \neq \varnothing, i = 1, 2, 3\}$
5: $B_4 \leftarrow [(a_1, x_2, x_3), (x_1 - 1, b_2, b_3)]$                // might be empty
6: $B_5 \leftarrow [(x_1, x_2, a_3), (b_1, b_2, x_3 - 1)]$                // might be empty
7: $B_6 \leftarrow [(x_1, a_2, a_3), (b_1, x_2 - 1, b_3)]$                // might be empty
8: $L \leftarrow L \cup \{B_i \mid B_i \neq \varnothing, i = 4, 5, 6\}$
```

Algorithm 4 and in Figure 4, splits the box $B = [a, b]$ into eight disjoint regions. (Here we make use of the integrality assumption – otherwise the regions would intersect in their boundaries.) Since x is a nondominated point, two of these regions, the boxes $[a, x]$ and $[x, b]$, are not interesting any longer, since they cannot contain any more nondominated points, cf. Figure 4. Of the other six, the three described in lines 1–3 in Algorithm 4 might still contain nondominated points, and so they become new high-priority boxes. The remaining three, those in lines 5–7 of Algorithm 4 might also contain new nondominated points, but large parts of them may be dominated by points in B_1, B_2 or B_3; hence, to reduce the search space, their examination is postponed until after the high-priority boxes have been completely processed, and so they become new low-priority boxes. In total, this inserts up to 3 new high-priority boxes into H and up to 3 new low-priority boxes into L; if the point x happens to lie on the boundary of B, some of the newly created boxes may be empty. Note that the *pop* function returns and

removes an element of the given list, so that the examined box is not in the list any more after the iteration is finished.
When no high-priority boxes are left, a low-priority box B is considered (line 15). This box might be partly or even completely dominated by a nondominated solution found earlier in one of the high-priority boxes, which is determined by Algorithm 5. If no such

Algorithm 5 *bestdominating*

```
Input: box $B = [a, b]$, set $S$ of objective points
Output: objective point dominating (part of) $B$ (possibly null)
1: if $\nexists x \in S : x \leqslant b$ then
2:     return null
3: else if $\exists x \in S : x \leqslant a$ then
4:     return $x$
5: else
6:     return any $x \in S$ with $x \leqslant b$          // preferably the "best" such point
7: end if
```

point can be found then B might still contain a nondominated point and consequently is upgraded to high priority (line 18), to be examined in the next iteration. Otherwise a point $\bar{x}$ is returned by Algorithm 5, which either completely dominates B, so that B can be skipped (line 20), or partly dominates B. In the last case, the part of B that is dominated need not be considered further and can be sliced off, which is done by Algorithm 6. This adds either 1 or 3 new low-priority boxes to the list L, depending on

Algorithm 6 *slicebox*

```
Input: box $B = [a, b]$, point $x \leqslant b$, list $L$ of low-priority boxes
Output: — (adds up to 3 new boxes to the list $L$)
1: depending on the position of $x$ relative to $B$, create new boxes $B_1, B_2, B_3$
2: $L \leftarrow L \cup \{B_i \mid B_i \neq \emptyset, i = 1, 2, 3\}$
```

the position of $\bar{x}$ relative to B, see Figure 5.

Remarks. As in Algorithm 1, *weightedsumsolve* and *lexminsolve* are exchangeable. The lower bounds in the call to *weightedsumsolve* are in fact not necessary; it is easy to see that, given any high-priority box B retrieved in line 8, there can be no nondominated point in the region in front of B (in any coordinate direction). However, the lower bounds of the boxes can result in lower bounds on the objective of the single-objective IP to solve, and hence speed up the execution of *weightedsumsolve*.
As with the Epsilon-constraint method, various notions of "best" in Algorithm 5 might make sense. Similarly, for the function *pop*(X), which returns (and removes) an element of the list X, various strategies might be devised; it does not necessarily have to return the first box in the list.

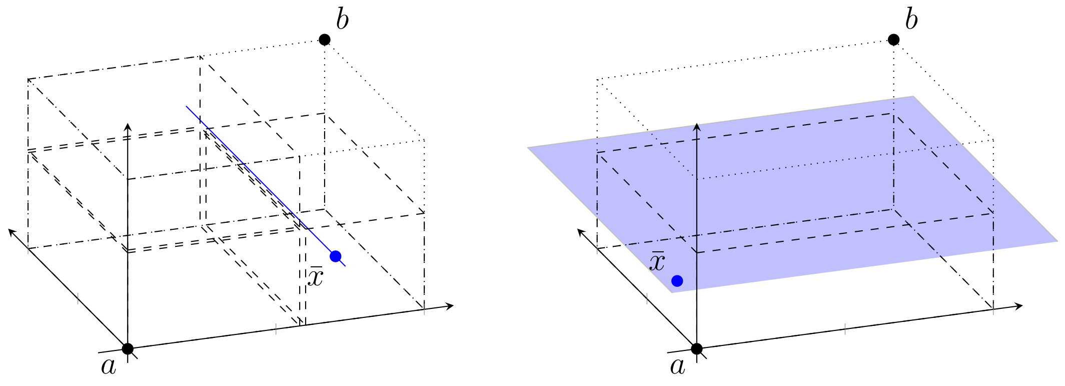


Figure 5: Two possibilities to slice a low-priority box $[a, b]$ yielding 3 or 1, respectively, new low-priority boxes (dashed)

4 Exemplary computations

The two algorithms were tested on a set of 10 instances, one of them a realistic test instance out of the set used in [7]. Table 1 shows the sizes of the instances and the resulting solution as well as the time spent by the two algorithms to compute the complete nondominated set. On the biggest instance, the Box-splitting method terminated for unknown reasons, possibly due to memory shortage during one IP-solving run, after the computation of 8261 out of 10463 nondominated solutions. In Figure 6, the solution time of both algorithms is plotted against the size of the nondominated set. On the considered instances, both algorithms perform similarly, with the Epsilon-constraint method tending to be faster on larger instances. Overall, the time seems to depend polynomially on the number of nondominated solutions. (The rightmost top point for the Box-splitting

*Approximate total time spent by the IP solver after finding 8261 nondominated solutions

$\lvert V\rvert$	$\lvert A\rvert$	$\lvert C\rvert$	$\lvert F\rvert$	nondom. solutions	ε-constraint	box-splitting
14	30	4	7	21	1.26	0.08
27	51	12	13	91	0.76	0.75
34	100	10	13	95	5.97	13.8
39	77	16	18	233	10.8	15.0
48	103	16	19	362	85.9	77.3
53	109	21	23	431	89.6	92.0
84	260	26	31	1564	12675	19899
96	295	31	37	2211	47003	73976
116	369	36	44	3285	323819	365682
424	1208	39	55	10463	1372050	>1594899*

Table 1: Sizes of instances, nondominated sets and runtimes of the two algorithms

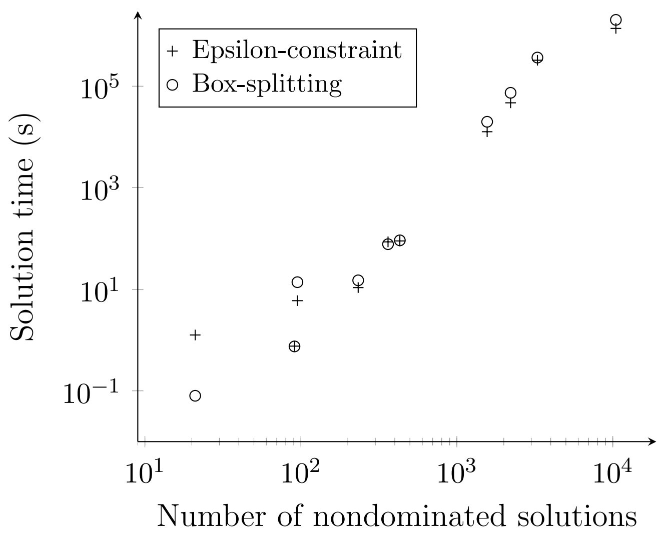


Figure 6: Time spent by the two methods depending on the size of the nondominated set (double logarithmic scale)

method results from a somewhat crude estimation by linearly extrapolating the time spent by the IP solver until the preliminary termination of the computation.)
Table 2 lists the runtimes of both methods for different choices of the solving subroutine: *lexminsolve*, *weightedsumsolve* with unit weights $(1, 1, 1)$ and with the weight

†8 solutions out of 3285 were failed to be identified, probably due to numerical inaccuracies of the IP solver

‡375373 seconds were spent on one single call to *lexminsolve*

§Out of memory

	ε-constraint		box-splitting		
$\lvert V\rvert$	unit weights	lexmin	unit weights	diag weights	lexmin
14	0.12	1.26	0.08	0.08	0.17
27	0.39	0.76	0.75	0.82	1.35
34	8.43	5.97	13.8	14.8	15.1
39	22.1	10.8	15.0	16.2	12.9
48	69.4	85.9	77.3	79.3	72.0
53	122.7	89.6	92.0	76.9	67.8
84	7414	12675	19899	17539	28681
96	30649	47003	73976	65694	425981‡
116	165762†	323819	365682	308630	—§

Table 2: Runtimes of the two methods with *weightedsumsolve* (employing unit weights $(1, 1, 1)$ and the diagonal direction in the box, respectively) and *lexminsolve* on the smaller instances

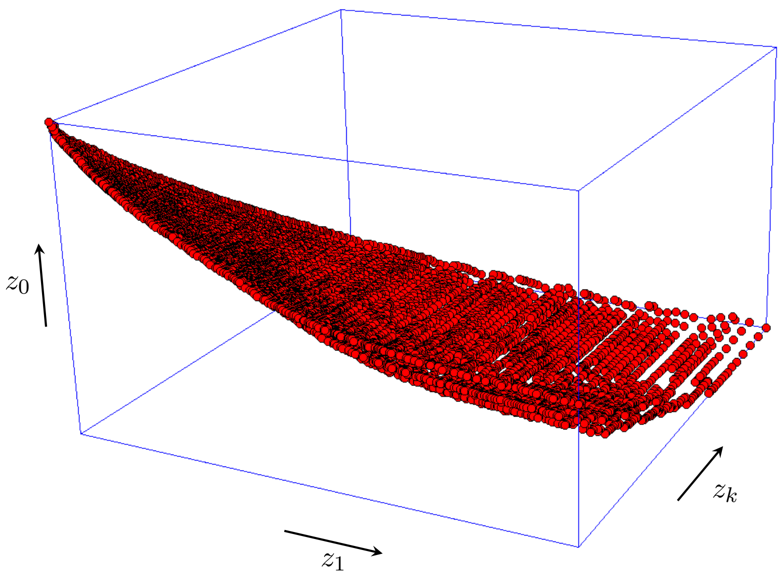


Figure 7: Exemplary set of nondominated solutions

vector corresponding to the diagonal in the current box, respectively, for the Box-splitting method. The *weightedsumsolve* variant apparently outperforms *lexminsolve* with increasing instance size and diagonal weights seem to work slightly better for the Box-splitting method. For one extremely long run of the Box-splitting method with *lexminsolve*, the solution of a single IP took up 88 % of the total computing time.
Hence, the biggest weakness of both methods appears to be the dependence on single IP-solving calls, which might take extremely long or consume huge amounts of memory in singular cases. This is more likely to happen, the more nondominated points have to be computed. A way to overcome this drawback would be to restrict resources for each IP-solving subroutine and try to control the approximation error made that way.
Figure 7 illustrates the nondominated set of the largest instance.

5 Extension to multiobjective problem

The 3-objective k-ArchConFL model extends naturally to a multiobjective model by taking the coverage with all considered architectures as objectives into account:

$$\begin{aligned}
\min \quad & z_\lambda := D - \sum_{l=1}^{\lambda} \sum_{j \in C} d_j z_j^l && \forall \lambda = 1, \dots, k \\
& z_0 := \sum_{(i,j) \in A_{rc}} c_{ij} x_{ij} + \sum_{l=1}^{k} \sum_{(i,j) \in A^l} c_{ij}^l x_{ij}^l + \sum_{l=1}^{k} \sum_{i \in F^l} c_i^l y_i^l \\
\text{s.t.} \quad & (1) - (6)
\end{aligned}$$

For k architectures, this yields a $(k+1)$-objective binary IP model.
Similarly, the two given algorithms can be extended to solve such multiobjective problems. For the Epsilon-constraint method, this amounts to enclosing the main loop into further outer loops corresponding to the extra objective dimensions – which can also be done in a recursive fashion – and extending the validity range to a k-tuple. For the Box-splitting method, the generalization is less straightforward; each $(k+1)$-dimensional box is split into 2^{k+1} new boxes, of which 2 again can be skipped, $k+1$ become new high-priority boxes, and the remaining boxes might be partly or fully dominated by points in a high-priority box and hence are given low priority initially.

References

[1] Vira Chankong and Yacov Y. Haimes. *Multiobjective decision making: theory and methodology.* Number 8 in Systems Sciences and Engineering Series. North-Holland, 1983.

[2] Kerstin Dächert. *Adaptive Parametric Scalarizations in Multicriteria Optimization.* PhD thesis, Bergische Universität Wuppertal, 2014.

[3] Kerstin Dächert and Kathrin Klamroth. A linear bound on the number of scalarizations needed to solve discrete tricriteria optimization problems. *Journal of Global Optimization*, 61(4):643–676, 2015.

[4] Matthias Ehrgott. *Multicriteria Optimization.* Springer, 2nd edition, 2005.

[5] Martin Grötschel, Christian Raack, and Axel Werner. Towards optimizing the deployment of optical access networks. *EURO Journal on Computational Optimization*, 2:17–53, 2014.

[6] Marco Laumanns, Lothar Thiele, and Eckart Zitzler. An efficient, adaptive parameter variation scheme for metaheuristics based on the epsilon-constraint method. *European Journal of Operational Research*, 169(3):932–942, 2006.

[7] Markus Leitner, Ivana Ljubić, Markus Sinnl, and Axel Werner. On the two-architecture connected facility location problem. *Electronic Notes in Discrete Mathematics*, 41:359–366, 2013.

[8] Markus Leitner, Ivana Ljubić, Markus Sinnl, and Axel Werner. A new exact method and matheuristics for bi-objective 0/1 ILPs: Application to FTTx-network design, 2015. Preprint Optimization Online: `http://www.optimization-online.org/DB_HTML/2015/03/4844.html`.