Computing an approximation of the nondominated set of multi-objective mixed-integer nonlinear optimization problems

Gabriele Eichfelder*, Leo Warnow*

November 30, 2023

Abstract

In practical applications, one often has not only one, but several objectives that need to be optimized simultaneously. What is more, modeling such real world problems usually involves using both, continuous and integer variables. This then results in multi-objective mixed-integer optimization problems, which are in focus of this paper. We present an approximation concept, called enclosure, for the nondominated set of such optimization problems and discuss how this concept arises as a natural extension of approximation concepts from single-objective global optimization. Further, we show how to practically compute such an enclosure for multi-objective mixed-integer convex optimization problems using the Hybrid Patch Decomposition algorithm (HyPaD). We demonstrate how exploiting the structure of this kind of optimization problems allows the HyPaD algorithm to operate almost entirely in the criterion space. This is not only in contrast to most algorithms for multi-objective nonconvex optimization problems, but also makes the algorithm's performance, at least to some extent, independent of the number of optimization variables. Moreover, we exploit the gap between theoretical and practical performance evaluation of solution algorithms, especially with regard to the HyPaD algorithm. More precisely, we present several realizations of the procedures that have originally been treated as black boxes in the theoretical evaluation of the algorithm. Finally, we provide numerical results for selected test instances. More precisely, we evaluate the different realizations of the black box procedures in terms of computation time, provide insights on the influence of the quality parameter chosen for the enclosure, and compare the HyPaD algorithm as a criterion space based method to the MOMIX algorithm as a decision space based method.

Key Words: multi-objective optimization, mixed-integer optimization, global optimization, approximation algorithm, enclosure

Mathematics subject classifications (MSC 2010): 90C11, 90C26, 90C29

*Institute of Mathematics, Technische Universität Ilmenau, Po 10 05 65, D-98684 Ilmenau, Germany, {gabriele.eichfelder,leo.warnow}@tu-ilmenau.de

1 Introduction

In this paper, we focus on a special class of multi-objective nonconvex optimization problems. More precisely, we consider multi-objective mixed-integer optimization problems where the nonconvexities arise due to integrality constraints on some of the variables. These optimization problems can be formally described as

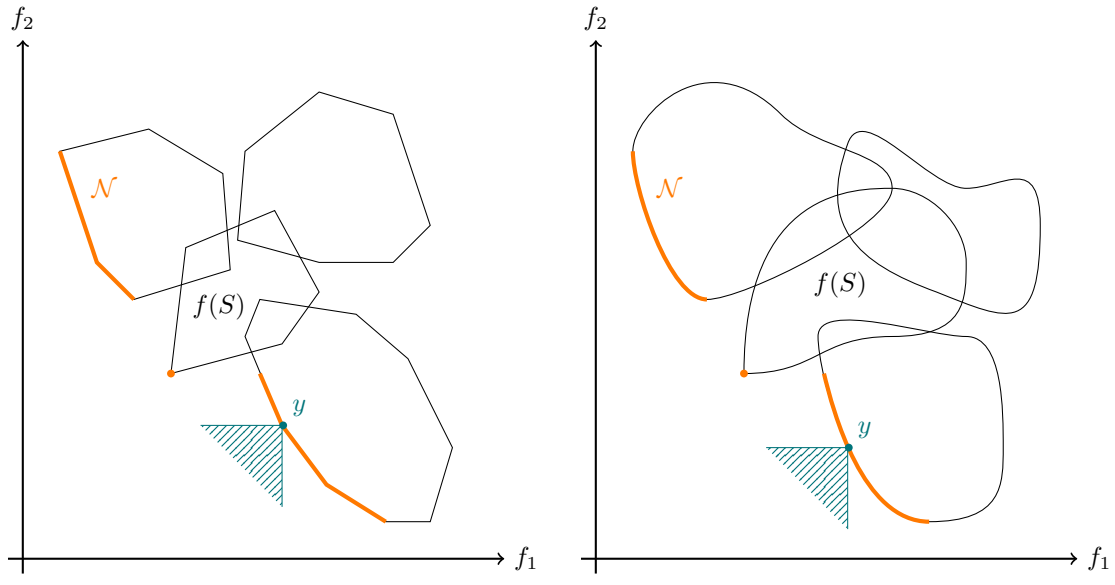
$$\begin{aligned} \min_{x} \quad & f(x) \\ \text{s.t.} \quad & g(x) \leq 0_q, \\ & x \in X := X_C \times X_I \end{aligned} \tag{MOMIP}$$

where $f = (f_1, \ldots, f_p) \colon \mathbb{R}^{n+m} \to \mathbb{R}^p$ denotes a vector of $p \in \mathbb{N}$ objective functions $f_i \colon \mathbb{R}^{n+m} \to \mathbb{R}$, $i \in [p]$ and $g = (g_1, \ldots, g_q) \colon \mathbb{R}^{n+m} \to \mathbb{R}^q$ denotes a vector of $q \in \mathbb{N}$ inequality constraint functions $g_j \colon \mathbb{R}^{n+m} \to \mathbb{R}$, $j \in [q]$ with $[n] := \{1, \ldots, n\}$ representing the set of the first $n \in \mathbb{N}$ natural numbers. Further, $0_q \in \mathbb{R}^q$ denotes the all-zeros vector, $X_C := [l_C, u_C] \subseteq \mathbb{R}^n$ denotes a nonempty box with $l_C, u_C \in \mathbb{R}^n$, and $X_I := [l_I, u_I] \cap \mathbb{Z}^m$ denotes a (finite) nonempty subset of $\mathbb{Z}^m$ with $l_I, u_I \in \mathbb{Z}^m$. We write $x = (x_C, x_I)$ for all $x \in X$ to distinguish between the continuous and integer variables of our optimization problem (MOMIP). The feasible set of (MOMIP) is denoted by S and is assumed to be nonempty.

There exists a large variety of practical applications that can be modeled as multi-objective mixed-integer optimization problems. The multi-objective aspect arises due to the different and often conflicting goals that a decision maker wants to achieve. A classic example for this is portfolio optimization where one wants to maximize the expected returns while simultaneously minimizing the risk of the investment. Since usually only some of the decision variables represent continuous parameters while others are discrete, many of these optimization problems are also mixed-integer. Let us consider portfolio optimization again. Several assets, for instance public stocks and (crypto-)currencies, can be bought and sold in any fraction. Thus, these can be modeled using continuous variables. Other assets, like real estate and physical products (e.g., luxury watches, cars, wine), can only be included in a portfolio in discrete amounts. Hence, these can be modeled as integer variables and the overall problem becomes a multi-objective mixed-integer optimization problem. Besides portfolio optimization, see for instance [48], there are several other classes of application problems that can be modeled as (MOMIP). These include medical applications [23, 44], supply chain problems such as water distribution and allocation [40, 47], and industrial applications [49] to only name a few.

As mentioned above, the $p > 2$ objective functions that are minimized simultaneously in (MOMIP) are usually conflicting. This means that, in general, there exists no feasible point $x \in S$ that minimizes all of them at the same time. In particular, there exists no unique optimal value as in single-objective optimization. Thus, a new optimality concept is needed. The concept that we focus on in this paper and which is used most of the time in multi-objective optimization is the concept of nondominance. A point $y \in f(S)$ is called a nondominated point of (MOMIP) if one cannot improve in one objective without deteriorating another. This means that for any $y' \in f(S)$ with $y' \leq y$, i.e., $y'_i \leq y_i$ for all $i \in [p]$, it must already hold $y' = y$. We denote by $\mathcal{N} \subseteq f(S)$ the set of all nondominated points of (MOMIP). An illustration of this concept is provided in Figure 1.

Since, in general, there exists an infinite number of nondominated points of (MOMIP), solving this optimization problem usually means computing an approximation of the nondominated set (or the so-called efficient set $\mathcal{E} \subseteq \mathbb{R}^{n+m}$, which is its decision space equivalent). There exist mainly two categories of corresponding approximation approaches in multi-objective optimization: representation approaches and coverage approaches. The output of representation approaches is a finite subset of the nondominated set, whereas coverage approaches compute a superset of the nondominated set. We provide a more detailed overview and comparison of both approaches in Section 2. While there exists a long track of research for multi-objective (continuous) optimization problems, see also the surveys [14, 42], this is not the case for multi-objective mixed-integer optimization problems. The first algorithm that explicitly exploits the mixed-integer structure of (MOMIP) was presented in 1998 by Mavrotas and Diakoulaki [33]. This branch-and-bound algorithm, as well as an improved version presented in 2005 [34], focuses on multi-objective mixed-integer linear optimization problems. These are optimization problems (MOMIP) where all objective and constraint functions are assumed to be linear. In fact, most of the early literature regarding multi-objective mixed-integer optimization focuses on that linear case. The reason for this is presumably that the nondominated set of such multi-objective mixed-integer linear optimization problems can be computed exactly. For instance, in the bi-objective setting, the nondominated set basically consists only of isolated points and line segments, where the latter can be fully represented by their end points as well. For an illustrative example, see Figure 1 (a). This property is exploited by many of the corresponding solution algorithms. For bi-objective mixed-integer linear optimization problems these include the Triangle Splitting Method [2] and also the more recent Boxed Line Method [37]. Also the algorithms from [1, 45, 46] address specifically bi-objective mixed-integer linear optimization problems.



(a) Bi-objective mixed-integer linear optimization problem

(b) Bi-objective mixed-integer nonlinear optimization problem

Figure 1: A nondominated point $y \in \mathbb{R}^2$ (highlighted in teal) and the nondominated set $\mathcal{N} \subseteq \mathbb{R}^2$ (highlighted in orange) for a bi-objective mixed-integer linear and nonlinear optimization problem

Already for multi-objective mixed-integer linear optimization problems with an arbitrary number of objective functions there exists noticeably less literature. In particular, most algorithms, like those from [38, 51] do no longer compute the exact nondominated set. Instead, they focus on the subset of so-called supported nondominated extreme points. One exception to this is the GoNDEF algorithm [39] presented in 2019. We refer to [28] for a more exhaustive survey on solution algorithms for multi-objective mixed-integer linear optimization problems.

In this paper, we focus on multi-objective mixed-integer nonlinear optimization problems. These are optimization problems (MOMIP) where at least one of the objective or constraint functions is nonlinear. The most important difference of these problems compared to the linear ones is that the nondominated set can, in general, no longer be computed exactly. This holds already for bi-objective mixed-integer nonlinear optimization problems and in particular for (MOMIP) with an arbitrary number of objective functions. This is one of the reasons why, in general, it is not possible to transfer or generalize techniques from multi-objective mixed-integer linear optimization to multi-objective mixed-integer nonlinear optimization. Hence, new approaches are needed. For an illustration of the nondominated set of a bi-objective mixed-integer nonlinear optimization problem, see Figure 1 (b).

Most algorithms for multi-objective mixed-integer nonlinear optimization problems focus on the particularly interesting subclass of multi-objective mixed-integer convex optimization problems. This means that all objective and constraint functions of (MOMIP) are assumed to be convex. Hence, not only the mixed-integer structure, but also the convexity can be exploited by the corresponding solution algorithms.

The first algorithm that was able to solve multi-objective mixed-integer convex optimization problems with an arbitrary number of objective functions was the MOMIX algorithm presented 2020 in [8]. In fact, this was the first algorithm that specifically addressed multi-objective mixed-integer convex optimization problems at all. While in this paper the focus is on algorithms that compute a coverage of the nondominated set, the algorithm from [8] computes a coverage of the efficient set, which is its decision space equivalent. Nonetheless, at least for such instances of (MOMIP) with convex quadratic objective and constraint functions, a coverage of the nondominated set can be derived from the coverage of the efficient set, see [8, Theorem 3.13].

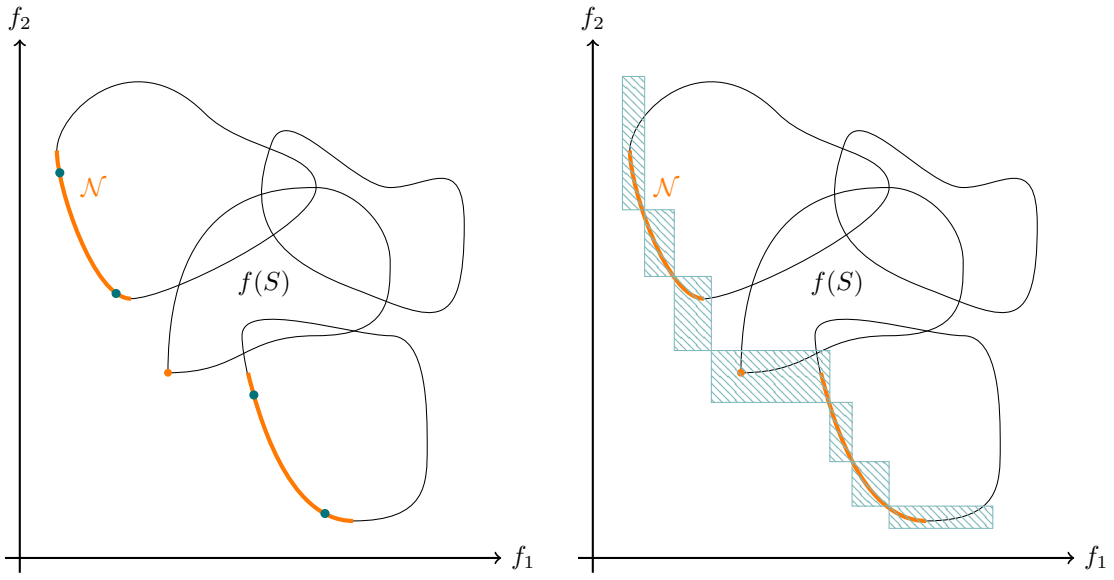
The first algorithms to directly compute a coverage of the nondominated set of multi-objective mixed-integer convex optimization problems, namely from [5] and [10], focus on the bi-objective setting. Since they rely heavily on the bi-objective structure, their techniques cannot directly be extended to multi-objective mixed-integer convex optimization problems with an arbitrary number of objective functions. The first algorithm to compute a coverage of the nondominated set of multi-objective mixed-integer convex optimization problems with an arbitrary number of objective functions was the Hybrid Patch Decomposition algorithm (HyPaD) [22] presented in 2023. This is also the algorithm in focus of this paper, see Section 3.

For completeness, we want to mention that also first algorithms for multi-objective mixed-integer nonconvex optimization problems exist. These include [31] for quadratically constrained problems and [18] for arbitrary multi-objective mixed-integer optimization problems.

The remaining part of this paper is structured as follows: In Section 2, we provide a brief overview of the different approximation concepts for the nondominated set in multi-objective optimization. We discuss how the so-called enclosure, a special kind of coverage, arises as a natural extension of approximation concepts from single-objective optimization and provide one method to compute it. We then briefly present the HyPaD algorithm from [22] in Section 3. We explain how it exploits the specific structure of multi-objective mixed-integer convex optimization problems and how this allows the algorithm to rely solely on criterion space based techniques, which is in contrast to algorithms that simply consider (MOMIP) as a multi-objective nonconvex optimization problem. In Section 4, we demonstrate how to adjust the algorithm for practical use without losing any of its theoretically guaranteed properties. More precisely, we describe how to practically realize certain subroutines of the HyPaD algorithm that are mainly treated as black boxes in [22]. To underline the implications of the suggested adjustments, we conclude the paper with numerical results for different realizations of the HyPaD algorithm in Section 5. We also compare it against the decision space based MOMIX algorithm from [8].

2 Selecting an approximation concept

In general, it is not possible to compute the nondominated set of a multi-objective nonlinear optimization problem exactly. Consequently, solving such optimization problems usually means computing an approximation of the nondominated set of prescribed quality. This holds for multi-objective continuous as well as multi-objective mixed-integer optimization problems. There exist several approximation concepts in multi-objective optimization. They can be divided in two main categories: representation approaches and coverage approaches. Representation approaches aim to compute a finite subset of the nondominated set and coverage approaches aim to compute a superset of the nondominated set. Figure 2 provides an example for the output of both approaches for a bi-objective mixed-integer nonlinear optimization problem.
In this paper, we focus on computing a coverage of the nondominated set of (MOMIP) instead of a representation. One reason for this is that, especially in the mixed-integer setting, the quality of a representation is hard to evaluate. In the literature, various quality measures for representations have been presented, see [24] for a survey. A commonly used quality criterion is the distance between different elements of the representation. This includes, for instance, the concepts called coverage and uniformity presented in [43]. Especially for multi-objective mixed-integer optimization problems, there can be gaps and also isolated points in their nondominated set, see Figure 2 again. Hence, a representation with equal distance between adjacent elements is not necessarily a good approximation of the nondominated set. Moreover, it is hard to guarantee that a representation contains the isolated points of the nondominated set since these can usually only be obtained by solving subproblems for a very specific choice of parameters within the corresponding solution algorithms.



(a) A representation of the nondominated set consisting of four nondominated points (highlighted in teal)

(b) A coverage/enclosure of the nondominated set (shaded in teal)

Figure 2: Comparison of a representation and a coverage/enclosure of the nondominated set $\mathcal{N} \subseteq \mathbb{R}^2$ (highlighted in orange) of a bi-objective mixed-integer convex optimization problem

The above mentioned issues for gaps and isolated points in the nondominated set do not occur within coverage approaches. Since a coverage, by definition, is a superset of all nondominated points, isolated points are usually treated just like any other nondominated point. Hence, there is no need for a special technique to detect gaps or isolated points in the nondominated set. Another strong argument in favor of coverage approaches is that they are a very natural extension of approximation concepts used in single-objective global optimization. This holds especially for the concept of the so-called enclosure.

2.1 Enclosure

In order to select an approximation concept for the nondominated set of multi-objective mixed-integer optimization problems, we briefly recall what is done in single-objective global optimization, which includes single-objective mixed-integer optimization. The optimal value for this class of optimization problems is usually not computed exactly but approximately. In most cases, this means that the corresponding solution algorithms compute a lower bound $l \in \mathbb{R}$ and an upper bound $u \in \mathbb{R}$ for the optimal value $\bar{y} \in \mathbb{R}$ of the optimization problem such that $l \leq \bar{y} \leq u$. The gap $\varepsilon := u - l$ between these two bounds then usually determines the quality of the approximation. More precisely, for every feasible point x of the single-objective optimization problem with $l \leq f(x) \leq u$ it holds that $|f(x) - \bar{y}| \leq \varepsilon$. Thus, x is an ε-optimal solution of the optimization problem.
We remark that this kind of approximation approach for the optimal value indeed also appears in classic solution techniques from single-objective mixed-integer optimization.

This includes, for instance, the well-known outer approximation technique as presented in [3, 11, 25].
The condition $l \leq \bar{y} \leq u$ from above for the optimal value $\bar{y} \in \mathbb{R}$ of a single-objective optimization problem can be rewritten as $\{\bar{y}\} \subseteq (\{l\} + \mathbb{R}_+) \cap (\{u\} - \mathbb{R}_+)$. This can be naturally generalized for our multi-objective setting as

$$\mathcal{N} \subseteq (L + \mathbb{R}^p_+) \cap (U - \mathbb{R}^p_+) \tag{2.1}$$

where $L, U \subseteq \mathbb{R}^p$ denote two nonempty sets. This immediately leads to the concept of a so-called enclosure of the nondominated set $\mathcal{N} \subseteq \mathbb{R}^p$ of (MOMIP) as presented in [17]. We use here a slightly modified and extended version of this concept.

Definition 2.1 *Let $L, U \subseteq \mathbb{R}^p$ be two nonempty and finite sets. Then the set*

$$\mathcal{C} = \mathcal{C}(L, U) := (L + \mathbb{R}^p_+) \cap (U - \mathbb{R}^p_+) = \bigcup_{l \in L} \bigcup_{u \in U} [l, u]$$

is called (box) coverage given L and U. If it holds that $\mathcal{N} \subseteq \mathcal{C}$ then we call $\mathcal{C} \subseteq \mathbb{R}^p$ an enclosure of $\mathcal{N} \subseteq \mathbb{R}^p$ with lower bound set $L \subseteq \mathbb{R}^p$ and upper bound set $U \subseteq \mathbb{R}^p$.

Within Definition 2.1, and within this paper in general, $[l, u] := (\{l\} + \mathbb{R}^p_+) \cap (\{u\} - \mathbb{R}^p_+)$ denotes the closed box (also called p-dimensional interval) with lower bound $l \in \mathbb{R}^p$ and upper bound $u \in \mathbb{R}^p$. For $l, u \in \mathbb{R}^p$ with $l_i > u_i$ for some index $i \in [p]$ we consequently obtain $[l, u] = \emptyset$. While allowing $l \not\leq u$ might seem counterintuitive at this point, it will be beneficial later on.
So far, we have generalized the approximation concept from single-objective global optimization to multi-objective global optimization. Next, we provide a generalization of the corresponding quality measure. Recall that in single-objective optimization the quality of an approximation of the optimal value $\bar{y} \in \mathbb{R}$ is determined by the gap $\varepsilon := u - l$. This is a suitable quality measure, since any feasible point $x \in S$ of the single-objective optimization problem with $y = f(x) \in [l, u]$ is ε-optimal. Hence, a natural extension of this quality concept for enclosures $\mathcal{C} \subseteq \mathbb{R}^p$ of the nondominated set $\mathcal{N} \subseteq \mathbb{R}^p$ of (MOMIP) should ensure that any attainable point $y \in f(S)$ with $y \in \mathcal{C}$ is ε-nondominated with $\varepsilon > 0$ representing the quality of the enclosure.
Let $L, U \subseteq \mathbb{R}^p$ be two finite and nonempty sets such that $\mathcal{C}(L, U) \subseteq \mathbb{R}^p$ is an enclosure of the nondominated set of (MOMIP). Further, recall that $y \in f(S)$ is called ε-nondominated if there exists no $y' \in f(S)$ with $y' \leq y - \varepsilon e$ and $y' \neq y - \varepsilon e$, where $e \in \mathbb{R}^p$ denotes the all-ones vector. Then $y - \varepsilon e \notin \mathcal{C}(L, U)$ is a sufficient criterion for $y \in f(S) \cap \mathcal{C}(L, U)$ to be ε-nondominated. Thus, we can define the quality of the enclosure $\mathcal{C}(L, U) \subseteq \mathbb{R}^p$ to be the largest $\varepsilon > 0$ such that there exists $y \in \mathcal{C}(L, U)$ with $y - \varepsilon e \in \mathcal{C}(L, U)$. In fact, this is exactly the definition of the width $w(\mathcal{C}(L, U)) \in \mathbb{R}$ of a nonempty enclosure $\mathcal{C}(L, U) \subseteq \mathbb{R}^p$ which was introduced and proposed as a quality measure in [17]. More precisely, the width of an enclosure $\mathcal{C}(L, U) \neq \emptyset$ is defined there as the optimal value of

$$\begin{aligned} \max_{y,t} \quad & \frac{\|(y + te) - y\|}{\sqrt{p}} \\ \text{s.t.} \quad & y \in \mathcal{C}(L, U), \\ & y + te \in \mathcal{C}(L, U), \\ & t \in \mathbb{R}_+. \end{aligned} \tag{2.2}$$

In [17, Lemma 3.1] it is also formally proven that, for any enclosure $\mathcal{C}(L, U) \subseteq \mathbb{R}^p$ of the nondominated set $\mathcal{N} \subseteq \mathbb{R}^p$ of (MOMIP), $w(\mathcal{C}(L, U)) < \varepsilon$ implies that all attainable points $y \in \mathcal{C}(L, U) \cap f(S)$ are indeed ε-nondominated.

As shown in [17, Lemma 3.2], there exists an equivalent formulation of (2.2) that can be evaluated only using the sets $L, U \subseteq \mathbb{R}^p$ of lower and upper bounds. What is more, this equivalent formulation also allows to extend the width concept to general coverages.

Definition 2.2 *Let $L, U \subseteq \mathbb{R}^p$ be two nonempty and finite sets. Then the width $w(\mathcal{C}(L,U)) \in \mathbb{R}$ of the (potentially empty) coverage $\mathcal{C}(L,U) \subseteq \mathbb{R}^p$ is defined as the optimal value of*

$$\begin{aligned} \max_{l,u} \quad & s(l,u) \\ \text{s.t.} \quad & l \in L,\ u \in U \end{aligned} \tag{2.3}$$

where $s(l,u) := \min_{i \in [p]}(u_i - l_i)$ denotes the shortest edge of the box $[l,u] \subseteq \mathbb{R}^p$.

We remark that the optimal value $w(\mathcal{C}(L,U)) \in \mathbb{R}$ of (2.3), and hence the width of the coverage, always exists. This is because, by definition, the sets $L, U \subseteq \mathbb{R}^p$ are assumed to be nonempty and finite.
At first, it might seem surprising that one needs to make use of the shortest edge length in (2.3) and not of the largest. However, this does not contradict the well-known quality concept of the largest gap between lower and upper bounds from single-objective global optimization. In particular, in case $p = 1$ the shortest edge length $s(l,u) = u - l$ is just the usual gap between the lower bound $l \in \mathbb{R}$ and the upper bound $u \in \mathbb{R}$. Consequently, the width would then denote the largest gap between all lower and upper bounds, i.e., $w(\mathcal{C}(L,U)) = \max\{u - l \mid l \in L, u \in U\}$. Since in single-objective global optimization there usually exists only a single upper bound $u \in U := \{u\}$, this equals the well-known quality measure $w(\mathcal{C}(L,U)) = \max\{u - l \mid l \in L\} = u - \min\{l \in L\}$ where $L \subseteq \mathbb{R}$ denotes some set of partial lower bounds, e.g., corresponding to subproblems obtained by a branch-and-bound scheme.
Finally, we remark that the shortest edge length and the width allow us to detect empty boxes and coverages. More precisely, for boxes $[l,u] \subseteq \mathbb{R}^p$ we have $s(l,u) < 0$ in case $l \not\leq u$ and conversely $s(l,u) \geq 0$ in case $l \leq u$. Hence, the shortest edge length allows us to detect empty boxes. Analogously, since $[l,u] = \emptyset$ if and only if $s(l,u) < 0$, we obtain that $\mathcal{C}(L,U) = \emptyset$ if and only if $w(\mathcal{C}(L,U)) < 0$. So the width can be used to detect empty coverages. This property is particularly useful for developing an algorithmically tractable discarding strategy.

2.2 Computation of lower and upper bounds

All the properties presented and discussed in Section 2.1 hold for arbitrary coverages $\mathcal{C}(L,U)$ as long as $L, U \subseteq \mathbb{R}^p$ are nonempty and finite sets. At the same time, a coverage is fully determined by these two sets. This means that the computation of a coverage, and in particular of an enclosure $\mathcal{C}(L,U)$ of the nondominated set of (MOMIP), is fully determined by the computation of the lower and upper bound sets.
Recall that several solution algorithms for single-objective global optimization problems make use of a single upper bound. This upper bound typically represents the best value $f(x) \in \mathbb{R}$ for some feasible point $x \in S$ that has been found so far. Additionally, they use a set $L \subseteq \mathbb{R}$ of lower bounds that consists of underestimators of the objective function on certain subsets $S' \subseteq S$, for instance obtained by a branch-and-bound scheme. In this section, we present a generalization of this idea for the multi-objective setting and in particular for (MOMIP).

Let $\bar{y} \in \mathbb{R}$ be the unique optimal value of a single-objective global optimization problem. Then for every feasible point $x \in S$ of that same problem we have that $\bar{y} \leq f(x)$ and hence $f(x)$ is a valid global upper bound on the optimal value. We can rewrite this as

$$\bar{y} \in \{f(x)\} - \mathbb{R}_+. \tag{2.4}$$

We can also rewrite $\bar{y} \leq f(x)$ as $\bar{y} \not> f(x)$ which is the same as

$$\bar{y} \notin \{f(x)\} + \operatorname{int}(\mathbb{R}_+). \tag{2.5}$$

For multi-objective optimization problems there no longer exists a unique optimal value, but an (in general) infinite set $\mathcal{N} \subseteq \mathbb{R}^p$ of nondominated points. Nonetheless, we still have that for any nondominated point $\bar{y} \in \mathcal{N}$ and for any attainable point $f(x) \in f(S) \subseteq \mathbb{R}^p$ it holds that

$$\bar{y} \notin \{f(x)\} + \operatorname{int}(\mathbb{R}^p_+) \tag{2.6}$$

which is a straightforward generalization of (2.5). However, this does not necessarily imply that $\bar{y} \in \{f(x)\} - \mathbb{R}^p_+$. Fortunately, there exists a method that, given a set $N \subseteq f(S)$, computes a set $U(N) \subseteq \mathbb{R}^p$ such that for every nondominated point $\bar{y} \in \mathcal{N}$ it holds that

$$\bar{y} \in U(N) - \mathbb{R}^p_+. \tag{2.7}$$

This is not only a proper generalization of (2.4), but also equivalent to $\mathcal{N} \subseteq U(N) - \mathbb{R}^p_+$ which is exactly the condition from Definition 2.1 that makes $U(N)$ a (global) upper bound set for an enclosure of the nondominated set of (MOMIP). This method is presented in [30] and introduces $U(N)$ as the so-called local upper bound set. In fact, some of these ideas already appeared in [13]. The concepts from [30] are used within a variety of literature. In particular, this includes literature related to so-called hyperboxing approaches or techniques that make use of the so-called hypervolume as a quality measure. This includes, for instance, the hyperboxing algorithm for multi-objective optimization problems from [12] as well as the approaches presented in [35] and [50]. We provide here not the classic definition of local upper bounds from [30], but a slightly generalized version based on [21, Definition 4.1].

Definition 2.3 *Denote by $A \subseteq \mathbb{R}^p$ an arbitrary area of interest and let $N \subseteq \mathbb{R}^p$ be a finite set. Then the lower search region for N is defined as*

$$s(N) := \{y \in \operatorname{int}(A) \mid y' \not\leq y \text{ for every } y' \in N\}$$

and the lower search zone for some $u \in \mathbb{R}^p$ is defined as

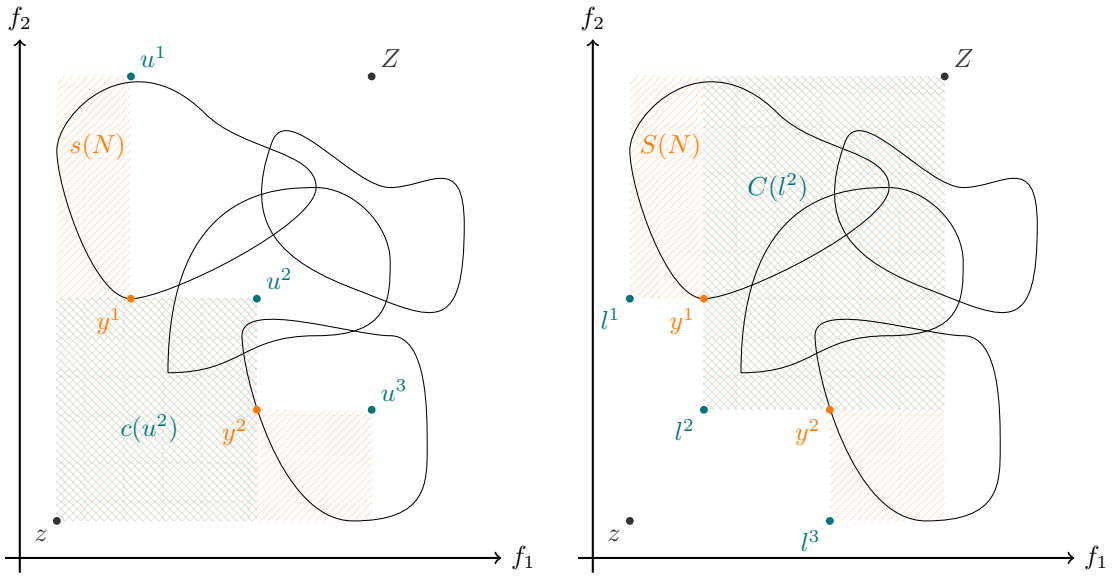
$$c(u) := \{y \in \operatorname{int}(A) \mid y < u\}.$$

A set $U = U(N) \subseteq \mathbb{R}^p$ is called local upper bound set given N if

(i) $s(N) = \bigcup_{u \in U(N)} c(u)$,

(ii) $\{u^1\} - \operatorname{int}(\mathbb{R}^p_+) \not\subseteq \{u^2\} - \operatorname{int}(\mathbb{R}^p_+)$ *for all* $u^1, u^2 \in U(N)$, $u^1 \neq u^2$.

Each point $u \in U(N)$ is called a local upper bound (LUB).



(a) A local upper bound set $U(N) = \{u^1, u^2, u^3\}$, the lower search region $s(N)$, and the lower search zone $c(u^2)$

(b) A local lower bound set $L(N) = \{l^1, l^2, l^3\}$, the upper search region $S(N)$, and the upper search zone $C(l^2)$

Figure 3: A local upper and local lower bound set for $N = \{y^1, y^2\}$ and the area of interest $A = [z, Z]$

For an in-depth discussion and comparison of this generalized version and the classic local upper bound concept from [30], we refer to [21]. An illustration of a local upper bound set is provided in Figure 3 (a).
In order for $U(N)$ to be an upper bound set for an enclosure of the nondominated set of (MOMIP), we need $N \subseteq \mathbb{R}^p$ to be a set of attainable points (or overestimators of such) and $A \subseteq \mathbb{R}^p$ to be an enclosure with $\mathcal{N} \subseteq \text{int}(A)$. The latter is more of a technical assumption that is needed for certain correctness proofs, see [21]. In case one is only interested in computing a coverage of a certain nonempty subset $\mathcal{N}' \subseteq \mathcal{N}$, it is possible to replace the assumption $\mathcal{N} \subseteq \text{int}(A)$ by $\mathcal{N}' \subseteq \text{int}(A)$. All concepts and results would then remain valid with respect to $\mathcal{N}'$.

Assumption 2.4 *The area of interest $A \subseteq \mathbb{R}^p$ is given as an enclosure*

$$A := \mathcal{C}(L', U') = (L' + \mathbb{R}^p_+) \cap (U' - \mathbb{R}^p_+)$$

where $L', U' \subseteq \mathbb{R}^p$ denote two finite sets of lower and upper bounds. Further, it holds that $\mathcal{N} \subseteq \text{int}(A)$.

We obtain the following result that formalizes the relation between local upper bound sets and upper bound sets for an enclosure of the nondominated set, see [21, Lemma 4.7].

Proposition 2.5 *Let $A \subseteq \mathbb{R}^p$ satisfy Assumption 2.4 and let $N \subseteq f(S) + \mathbb{R}^p_+$ be a finite set. Further, let $U(N) \subseteq \mathbb{R}^p$ be some finite local upper bound set. Then $U(N)$ is an upper bound set for an enclosure of the nondominated set of (MOMIP).*

While in [30] only the concept of (classic) local upper bounds was introduced, the idea can easily be transferred to also obtain a lower bounding concept. Consequently, the corresponding lower bound sets are called local lower bound sets, see for instance [19, Definition 3.4]. Again, we present here a slightly generalized version of this concept based on [21, Definition 4.2] and provide an illustration of it in Figure 3 (b).

Definition 2.6 *Denote by $A \subseteq \mathbb{R}^p$ an arbitrary area of interest and let $N \subseteq \mathbb{R}^p$ be a finite set. Then the upper search region for N is defined as*

$$S(N) := \{y \in \operatorname{int}(A) \mid y' \not\geq y \text{ for every } y' \in N\}$$

and the upper search zone for some $l \in \mathbb{R}^p$ is defined as

$$C(l) := \{y \in \operatorname{int}(A) \mid y > l\}.$$

A set $L = L(N) \subseteq \mathbb{R}^p$ is called local lower bound set given N if

(i) $S(N) = \bigcup_{l \in L(N)} C(l)$,

(ii) $\{l^1\} + \operatorname{int}(\mathbb{R}^p_+) \not\subseteq l^2 + \operatorname{int}(\mathbb{R}^p_+)$ *for all* $l^1, l^2 \in L(N)$, $l^1 \neq l^2$.

Each point $l \in L(N)$ is called a local lower bound (LLB).

The main question at this point is how to choose the set $N \subseteq \mathbb{R}^p$ such that a local lower bound set $L(N)$ is actually a lower bound set for an enclosure of the nondominated set of (MOMIP). Recall that in single-objective global optimization a global lower bound $l \in \mathbb{R}$ of the unique optimal value $\bar{y} \in \mathbb{R}$ needs to satisfy $l \leq \bar{y}$. We can rewrite this as $l \not> \bar{y}$ which is the same as

$$l \notin \{\bar{y}\} + \operatorname{int}(\mathbb{R}_+) \subseteq f(S) + \operatorname{int}(\mathbb{R}_+). \tag{2.8}$$

The multi-objective equivalent of the very same condition (2.8) is sufficient to obtain a relation between local lower bound sets and lower bound sets for an enclosure of the nondominated set of (MOMIP). Analogously to Proposition 2.5, we obtain the following result based on [21, Lemma 4.7].

Proposition 2.7 *Let $A \subseteq \mathbb{R}^p$ satisfy Assumption 2.4 and let $N \subseteq \mathbb{R}^p \setminus (f(S) + \operatorname{int}(\mathbb{R}^p_+))$ be a finite set. Further, let $L(N) \subseteq \mathbb{R}^p$ be some finite local lower bound set. Then $L(N)$ is a lower bound set for an enclosure of the nondominated set of (MOMIP).*

Finally, we can combine Propositions 2.5 and 2.7 in order to obtain a result that states how to obtain valid upper and lower bound sets for an enclosure of the nondominated set of (MOMIP), see also [21, Lemma 4.7].

Theorem 2.8 *Let $A \subseteq \mathbb{R}^p$ satisfy Assumption 2.4 and let*

$$N^1 \subseteq f(S) + \mathbb{R}^p_+ \text{ and}$$
$$N^2 \subseteq \mathbb{R}^p \setminus (f(S) + \operatorname{int}(\mathbb{R}^p_+))$$

be two finite sets. Further, let $U(N^1), L(N^2) \subseteq \mathbb{R}^p$ be some corresponding finite local upper and local lower bound sets. Then the coverage $\mathcal{C}(L(N^2), U(N^1)) \subseteq \mathbb{R}^p$ is an enclosure of the nondominated set $\mathcal{N}$ of (MOMIP) with upper bound set $U(N^1)$ and lower bound set $L(N^2)$.

3 The Hybrid Patch Decomposition algorithm

In this section, we present one particular algorithm, namely the Hybrid Patch Decomposition algorithm (HyPaD) from [22], for solving multi-objective mixed-integer convex optimization problems (MOMIP). Hence, from now on we assume all objective functions $f_i, i \in [p]$ and all constraint functions $g_j, j \in [q]$ to be once continuously differentiable and convex.

The HyPaD algorithm computes an enclosure of the nondominated set of (MOMIP) and uses the local upper and local lower bound concepts. What is more, it perfectly demonstrates the advantages of exploiting the structure of (MOMIP), i.e., its mixed-integer structure and the convexity of all objective and constraint functions. In particular, this allows the algorithm to perform its computations almost entirely in the criterion space. This is in contrast to most algorithms for general multi-objective nonconvex optimization problems that rely on at least some decision space based techniques, e.g., branch-and-bound, in order to handle the nonconvexities.

As its name already suggests, the HyPaD algorithm is based on a decomposition approach. More precisely, it decomposes the mixed-integer optimization problem (MOMIP) into several purely continuous subproblems

$$\begin{aligned} \min_{x_C} \quad & f(x_C, \hat{x}_I) \\ \text{s.t.} \quad & g(x_C, \hat{x}_I) \leq 0_q, \\ & x_C \in X_C \end{aligned} \tag{P($\hat{x}_I$)}$$

where $\hat{x}_I \in S_I := \{x_I \in \mathbb{Z}^m \mid \exists\, x_C \in \mathbb{R}^n \colon (x_C, x_I) \in S\}$ denotes some feasible integer assignment. These so-called patch problems (P($\hat{x}_I$)), also referred to as slice problems in [45], serve two purposes. First of all, they are used to compute attainable points of (MOMIP). A corresponding local upper bound set $U(N)$ then serves as the global upper bound set for the enclosure of the nondominated set of (MOMIP).

The attainable points are computed by solving the single-objective continuous convex optimization problem

$$\begin{aligned} \min_{x_C, t} \quad & t \\ \text{s.t.} \quad & f(x_C, \hat{x}_I) - l - t(u - l) \leq 0_p, \\ & g(x_C, \hat{x}_I) \leq 0_q, \\ & x_C \in X_C,\ t \in \mathbb{R} \end{aligned} \tag{SUP($\hat{x}_I, l, u$))}$$

where $[l, u] \subseteq \mathbb{R}^p$ is some box with $s(l, u) > \varepsilon > 0$. Let $(\bar{x}_C, \bar{t})$ be an optimal solution of (SUP($\hat{x}_I, l, u$)), which exists by [27, Proposition 2.3.4 and Theorem 2.3.1]. Then $y^1 := f(\bar{x}_C, \hat{x}_I) \in f(S)$ is not only an attainable point of (MOMIP), but also a weakly nondominated point of (P($\hat{x}_I$)) by [36, Theorem 3.2]. Hence, we have that $y^2 := l + \bar{t}(u - l) \notin f(S_{\hat{x}_I}) + \operatorname{int}(\mathbb{R}^p_+)$, where $S_{\hat{x}_I} = \{x \in S \mid x_I = \hat{x}_I\}$. This means that the patch problems do not only allow to compute and update a set $N^1 \subseteq f(S)$ for a global upper bound set $U := U(N)$, see Proposition 2.5. They also allow to compute and update a set $N^2 \subseteq \mathbb{R}^p \setminus (f(S_{\hat{x}_I}) + \operatorname{int}(\mathbb{R}^p_+))$ and hence a patch level lower bound set $L_{\hat{x}_I} := L(N)$ for an enclosure of the nondominated set of (P($\hat{x}_I$)), see Proposition 2.7.

In analogy to partial lower bounds in single-objective global optimization, these patch level lower bound sets could be used to obtain a global lower bound set for an enclosure of the nondominated set of (MOMIP) as

$$L := \bigcup_{\hat{x}_I \in \mathcal{E}_I} L_{\hat{x}_I} \tag{3.1}$$

where $\mathcal{E}_I := \{x_I \in X_I \mid \exists\, x_C \in X_C \colon (x_C, x_I) \in \mathcal{E}\} \subseteq S_I$ denotes the set of so-called efficient integer assignments. However, as already discussed in [22], such a lower bounding strategy would in practice enforce the HyPaD algorithm to enumerate all integer assignments $x_I \in X_I$. So while being a viable approach in theory, see also [22, Lemma 6.3], this strategy is only practically relevant in case of sets X_I of small cardinality.

The main lower bounding technique in the HyPaD algorithm works directly on a global level. More precisely, the global lower bound set L is computed by solving linear relaxations

$$\begin{aligned} \min_{x,\eta} \quad & \eta \\ \text{s.t.} \quad & f_i(\hat{x}) + \nabla f_i(\hat{x})^\top (x - \hat{x}) \leq \eta_i \quad \forall i \in [p] \quad \forall \hat{x} \in \mathcal{X}, \\ & g_j(\hat{x}) + \nabla g_j(\hat{x})^\top (x - \hat{x}) \leq 0 \quad \forall j \in [q] \quad \forall \hat{x} \in \mathcal{X}, \\ & x \in X,\ \eta \in \mathbb{R}^p \end{aligned} \tag{R($\mathcal{X}$)}$$

of (MOMIP), where $\mathcal{X} \subseteq \mathbb{R}^{n+m}$ denotes a finite and nonempty set of linearization points. Similarly to (SUP($\hat{x}_I, l, u$)) on the patch level, optimal solutions $(\bar{x}, \bar{\eta}, \bar{t})$ of the single-objective mixed-integer linear optimization problem

$$\begin{aligned} \min_{x,\eta,t} \quad & t \\ \text{s.t.} \quad & \eta - l - t(u - l) \leq 0_p, \\ & f_i(\hat{x}) + \nabla f_i(\hat{x})^\top (x - \hat{x}) \leq \eta_i \quad \forall i \in [p] \quad \forall \hat{x} \in \mathcal{X}, \\ & g_j(\hat{x}) + \nabla g_j(\hat{x})^\top (x - \hat{x}) \leq 0 \quad \forall j \in [q] \quad \forall \hat{x} \in \mathcal{X}, \\ & x \in X,\ \eta \in \mathbb{R}^p,\ t \in \mathbb{R} \end{aligned} \tag{RSUP($\mathcal{X}, l, u$))}$$

with $[l, u] \subseteq \mathbb{R}^p$ and $s(l, u) > 0$ yield a weakly nondominated point $\bar{\eta} \in \mathbb{R}^p$ of (R($\mathcal{X}$)). In particular, we have that $\bar{\eta} \notin f(S) + \operatorname{int}(\mathbb{R}^p_+)$ and hence these points can be used to compute and update a global lower bound set for an enclosure of the nondominated set of (MOMIP), see Proposition 2.7. We use here the term global to clearly indicate that the lower bound set L is a lower bound set for an enclosure of the nondominated set of (MOMIP). This allows us to distinguish it from the patch level lower bound sets $L_{\hat{x}_I}, \hat{x}_I \in S_I$ from above. For those, it only holds that $\mathcal{N}_{\hat{x}_I} \subseteq L_{\hat{x}_I} + \mathbb{R}^p_+$ and, in general, not that $\mathcal{N} \subseteq L_{\hat{x}_I} + \mathbb{R}^p_+$.

In the end, the HyPaD algorithm computes an enclosure of the nondominated set of (MOMIP) of prescribed width $\varepsilon > 0$ in finitely many steps, see [22, Theorems 6.5 and 6.7]. Thereby, it reduces solving the multi-objective mixed-integer convex optimization problem (MOMIP) to solving single-objective continuous optimization problems (SUP($\hat{x}_I, l, u$)) and single-objective mixed-integer linear optimization problems (RSUP($\mathcal{X}, l, u$)). For both these classes of optimization problems fast and reliable solvers are available. Beyond that, the HyPaD algorithm constantly alternates between updating the lower bounds on a global level by solving (RSUP($\mathcal{X}, l, u$)) and updating the global upper bound set as well as the patch level lower bound set for some $\hat{x}_I \in S_I$ by solving (SUP($\hat{x}_I, l, u$)). In particular, the optimal solutions of (RSUP($\mathcal{X}, l, u$)) determine the integer assignment $\hat{x}_I \in S_I$ for (SUP($\hat{x}_I, l, u$)).

This constant interplay of computations on the global level and on the patch level is the reason why the algorithm is called hybrid. Also, iteratively performing updates on the patch level allows the HyPaD algorithm to detect patches that do not contribute to the overall nondominated set of (MOMIP) early on and to spend as little computational effort as possible on these patches. In particular, usually only a small share of all possible patches (P($\hat{x}_I$)), $\hat{x}_I \in S_I$ is considered at all. This means that, in addition to not requiring a priori knowledge of the set S_I of feasible integer assignments, the HyPaD algorithm also needs to compute only a small subset of those, see also the numerical results in Section 5.3.
We provide the pseudocode of the HyPaD algorithm and all its subroutines in the appendix at the end of this paper. An open source implementation of the algorithm is publicly available on GitHub [20].

4 Implementation details

While theoretical analysis of solution algorithms, e.g., regarding finiteness, correctness, and complexity, is important, there often exist different criteria which are used by decision makers in practice to choose an algorithm for their particular problem. In particular, these end users are often not so much interested in the theoretical performance analysis of an algorithm, but its performance for their specific type of optimization problem. For instance, if one algorithm possesses a worse complexity bound than another, it could still be preferred by the user if its computation times for their particular optimization problem are better. Fortunately, most algorithms possess a certain degree of freedom that allows to modify the algorithm without loosing the theoretical results and guarantees. Hence, it is important to identify these degrees of freedom and to discuss possible modifications of the algorithm that could help to address the additional criteria of the decision maker.
In this section, we discuss two aspects of the HyPaD algorithm that fall into this category. They are mainly treated as a black box in the theoretical analysis in [22], but could play an important role for the practical application of the algorithm. The first aspect is the initialization of the overall algorithm as well as the initialization of the patches, more precisely of the corresponding entries of the integer data structure. The second aspect is the realization of the Search New Integer Assignment (SNIA) procedure. This is a fallback procedure that forces the HyPaD algorithm to compute a new integer assignment $x_I \in X_I$ in case all previously computed integer assignments are infeasible or inactive.

4.1 Initialization

For the initialization of the HyPaD algorithm (Algorithm 5), we need a starting point $\hat{x} \in X$ and an initial area of interest $A \subseteq \mathbb{R}^p$ that satisfies Assumption 2.4.
For consistency with [22] where the HyPaD algorithm has been introduced, we provide A manually and do not compute it within MATLAB for our numerical experiments in Section 5. More precisely, we choose it to be a box $A := [z, Z]$ with $z, Z \in \mathbb{R}^p$ such that $f(S) \subseteq \text{int}(A)$. However, if one wants to include a mechanism to compute such an initial box within MATLAB, then using interval arithmetic (for example via INTLAB [41]) would be a suitable approach to do that.

For certain subclasses of (MOMIP), there also exist more sophisticated approaches to compute an initial area of interest. An example for multi-objective mixed-integer quadratic optimization problems is presented in [9].
To obtain a starting point for Algorithm 5, we solve the following single-objective continuous convex optimization problem which is basically a scalarization of the integer relaxed formulation of (MOMIP):

$$\begin{array}{ll} \min\limits_{x,t} & t \\ \text{s.t.} & f(x) - z - t(Z - z) \leq 0_p, \\ & g(x) \leq 0_q, \\ & x \in X_C \times [l_I, u_I] \subseteq \mathbb{R}^{n+m},\ t \in \mathbb{R}. \end{array} \tag{P_{init}}$$

We denote by $(\bar{x}, \bar{t})$ an optimal solution of (P_{init}). Then we can split $\bar{x} = (\bar{x}_C, \bar{x}_I)$ into a first part with n components and a second part with m components. The starting point $\hat{x} \in X$ for Algorithm 5 is then obtained by rounding the last m components, i.e., $\hat{x} = (\bar{x}_C, \lfloor \bar{x}_I + 0.5e \rfloor) \in X$.
Besides the initialization of the overall HyPaD algorithm, we also need to initialize new entries of the integer data structure $\mathcal{D}$ whenever a new feasible integer assignment is computed. For any such feasible integer assignment $\hat{x}_I \in S_I$ the entry $\mathcal{D}(\hat{x}_I)$ of the integer data structure $\mathcal{D}$ is used within the HyPaD algorithm to collect and store data obtained for the patch problem (P($\hat{x}_I$)). More precisely, each entry $\mathcal{D}(\hat{x}_I)$ consists of four components. The first one is a set of lower bounds for the nondominated set $\mathcal{N}_{\hat{x}_I}$ of (P($\hat{x}_I$)), denoted by $\mathcal{D}(\hat{x}_I).L$. The second one is a boolean value $\mathcal{D}(\hat{x}_I).S$ that indicates whether further computations for the patch problem (P($\hat{x}_I$)) are needed. For example, this value is set to false in case it is recognized that $\mathcal{N}_{\hat{x}_I}$ does not contribute to the nondominated set $\mathcal{N}$ of (MOMIP), i.e., $\mathcal{N}_{\hat{x}_I} \cap \mathcal{N} = \emptyset$. We call the integer assignment $\hat{x}_I$ active if $\mathcal{D}(\hat{x}_I).S$ is set to true and inactive otherwise. All weakly efficient points $x \in S_{\hat{x}_I}$ of the subproblem (P($\hat{x}_I$)) computed by the HyPaD algorithm are saved in the set $\mathcal{D}(\hat{x}_I).E$. Analogously, the final component $\mathcal{D}(\hat{x}_I).N$ contains the weakly nondominated points $y \in \mathbb{R}^p$ of (P($\hat{x}_I$)) that are computed within the algorithm.
In [22] the initialization of $\mathcal{D}(\hat{x}_I)$ is only roughly outlined. Hence, we show how the first lower bound $z \in \mathbb{R}^p$ with $f(S_{\hat{x}_I}) \subseteq \{z\} + \operatorname{int}(\mathbb{R}^p_+)$ for the corresponding patch problem is computed in more detail in this section. We present here the exact method that we actually use for our numerical tests. This method is based on the computation of the ideal point corresponding to (P($\hat{x}_I$)). For $i \in [p]$ we consider the single-objective continuous convex optimization problem

$$\begin{array}{ll} \min\limits_{x_C} & f_i(x_C, \hat{x}_I) \\ \text{s.t.} & g(x) \leq 0_q, \\ & x_C \in X_C. \end{array} \tag{PI($\hat{x}_I, i$)}$$

Let $\bar{x}^i_C$ be an optimal solution of (PI($\hat{x}_I, i$)) and denote by $\bar{z}_i := f_i(\bar{x}^i_C, \hat{x}_I)$ the corresponding optimal value. Then $\bar{z} = (\bar{z}_1, \ldots, \bar{z}_p)$ is called the ideal point for the patch problem corresponding to the integer assignment $\hat{x}_I \in S_I$. By using a small offset $\sigma > 0$, this allows us to initialize the integer data structure $\mathcal{D}(\hat{x}_I)$ as shown in Algorithm 1.
One benefit of this particular initialization strategy is that it initializes not only $\mathcal{D}(\hat{x}_I).L$, but also $\mathcal{D}(\hat{x}_I).E$ and $\mathcal{D}(\hat{x}_I).N$. Consequently, not only the patch level lower bound set $\mathcal{D}(\hat{x}_I).L$ is initialized. We also obtain p attainable points of (MOMIP), contained in $\mathcal{D}(\hat{x}_I).N$, that can be used to update the global upper bound set U.

Algorithm 1 Initialization of $\mathcal{D}(\hat{x}_I)$ for a new integer assignment $\hat{x}_I \in S_I$

Input: New integer assignment $\hat{x}_I \in S_I$, offset $\sigma > 0$
Output: Initialized entry $\mathcal{D}(\hat{x}_I)$ of the integer data structure

1: **procedure** InitIDS($\hat{x}_I$)
2: For all $i \in [p]$ solve (PI($\hat{x}_I, i$)) with optimal solution $\bar{x}_C^i$ and optimal value $\bar{z}_i \in \mathbb{R}$
3: Compute $z \in f(S_{\hat{x}_I}) - \operatorname{int}(\mathbb{R}^p_+)$ with $z_i := \bar{z}_i - \sigma$ for all $i \in [p]$
4: Initialize $\mathcal{D}(\hat{x}_I).L = \{z\}$, $\mathcal{D}(\hat{x}_I).E = \{(\bar{x}_C^1, \hat{x}_I), \dots, (\bar{x}_C^p, \hat{x}_I)\}$, $\mathcal{D}(\hat{x}_I).N = \{f(\bar{x}_C^1, \hat{x}_I), \dots, f(\bar{x}_C^p, \hat{x}_I)\}$, $\mathcal{D}(\hat{x}_I).S = \text{true}$
5: **end procedure**

4.2 Fallback procedure for the computation of new integer assignments

The HyPaD algorithm is basically an interplay of computing integer assignments $x_I \in X_I$ by solving the single-objective mixed-integer linear problem (RSUP($\mathcal{X}, l, u$)) with $l, u \in \mathbb{R}^p, l < u, \emptyset \neq \mathcal{X} \subseteq \mathbb{R}^{n+m}$, and improving the coverages corresponding to patch problems (P($\hat{x}_I$)) for certain feasible integer assignments $\hat{x}_I \in S_I$. The coverages of all patches need to be improved only finitely often, see [22, Lemma 6.4]. This is mainly because after finitely many improvement steps, see Algorithm 7, we have that $w(L_{\hat{x}_I}, U) \leq \varepsilon$. Hence, it can happen that all integer assignments that have been computed by the HyPaD algorithm at a certain point are either infeasible or inactive, i.e., the corresponding coverages on the patch level need no further improvement. For the correctness of the overall algorithm, one needs to ensure that the algorithm will not get stuck in this situation. This can be achieved by forcing the HyPaD algorithm to compute a new integer assignment that has not been computed yet. If we denote by $\mathcal{X}$ a set such that

$$\mathcal{X}_I := \{x_I \in X_I \mid x = (x_C, x_I) \in \mathcal{X}\}$$

contains all the integer assignments that have already been explored, the Search New Integer Assignment procedure (SNIA), see Algorithm 2, computes such a new integer assignment. More precisely, this is done in line 2 of the algorithm and this is the only step that we consider in this section. For more details on the remaining steps and details of Algorithm 2, we refer to [22].
Concerning the theory presented in [22], it is not important how the new integer assignment is computed in line 2 of Algorithm 2 as long as this is done within a finite number of steps. However, in practice the method to compute a new integer assignment can play an important role, for example in terms of the overall computation time of the algorithm. In the following, we present and discuss three methods to realize line 2 of Algorithm 2. We make us of the assumption that X_I is finite and basically given as a box $X_I := [l_I, u_I] \cap \mathbb{Z}^m$ with $l_I, u_I \in \mathbb{Z}^m$. The total number of possible integer assignments is denoted by $k := |X_I|$.

Algorithm 2 Search new integer assignment

Input: Linearization points $\mathcal{X}$, integer data structure $\mathcal{D}$
Output: Updated set $\mathcal{X}$, integer data structure $\mathcal{D}$ (, bound sets L, U)

1: **procedure** SNIA($\mathcal{X}, \mathcal{D}$)
2: Search new $\tilde{x} \in X$ such that there exists no $x \in \mathcal{X}$ with $x_I = \tilde{x}_I$ and $\mathcal{D}(\tilde{x}_I)$ is not initialized
3: **if** no such $\tilde{x}$ exists **then**
4: Let $L := \{y \in \mathcal{D}.L \mid y \text{ is nondominated given } \mathcal{D}.L \text{ w.r.t } \leq\}$
5: Terminate **HyPaD** with output sets L, U
6: **else if** $\tilde{x}_I \in S_I$ **then**
7: InitIDS($\tilde{x}_I$)
8: **else**
9: Solve (F($\tilde{x}_I$)) with optimal solution $(\bar{x}_C, \bar{\alpha})$
10: Update linearization points: $\mathcal{X} = \mathcal{X} \cup \{(\bar{x}_C, \tilde{x}_I)\}$
11: **end if**
12: **end procedure**

4.2.1 Full enumeration

The first idea to discuss is a full enumeration of X_I. Since there exists a bijection between X_I and $[k]$ we can start with $i = 1$ and then count up to $i = k$ with each call of Algorithm 2. If the integer assignment belonging to $i \in [k]$ has already been computed, i.e., is contained in $\mathcal{X}_I$, then we just increment i further until we find an assignment of i that corresponds to a new integer assignment or $i > k$ which indicates that all integer assignments have already been computed and the algorithm can be terminated, see line 5 of Algorithm 2.
The full enumeration approach is the cheapest among the three methods presented in this paper in terms of computation time of Algorithm 2. This is mainly because this realization of the SNIA procedure avoids the computational overhead and effort of more advanced procedures like creating certain substructures in the decision space, see Sections 4.2.2 and 4.2.3. In particular, this makes full enumeration a very good choice for two scenarios. The first one are problems (MOMIP) where the number k of integer assignments is relatively small. Then a simple enumeration is just faster than any other strategy that introduces additional overhead. The second one are such problems (MOMIP) where promising candidates $x_I \in X_I$ for feasible integer assignments are known a priori. In that scenario, one could ensure that these candidates are explored first. However, such specific knowledge regarding the set $S_I \subseteq X_I$ is usually not given. In particular, it could be that the feasible integer assignments are explored last by the full enumeration approach. Especially if there exist only a few feasible integer assignments and a large number k of possible integer assignments, this could be an issue. For that reason, one might instead prefer approaches that are guaranteed to explore the decision space and hence the set X_I of integer assignments more evenly in order to increase the chance to find feasible integer assignments early on. In the following, we present two approaches that follow exactly this motivation.

4.2.2 Dynamic boxes

This approach is a branching technique and searches for a subbox of X_I that contains none of the visited integer assignments $x_I \in \mathcal{X}_I$, see Algorithm 3. Since we adapt the size of the considered boxes within the algorithm, we call this approach the dynamic boxes approach. For that algorithm we assume that $|\mathcal{X}_I| < k$, which can be checked beforehand.

Algorithm 3 Computing a new integer assignment by finding an empty subbox of X_I

Input: Initial box $X_I := [l_I, u_I]$, set of visited integer assignments $\mathcal{X}_I$
Output: New integer assignment $\hat{x}_I \in X_I \setminus \mathcal{X}_I$

procedure DynamicSNIA($X_I, \mathcal{X}_I$)
  while true **do**
    Compute edge lengths $w = u_I - l_I$ and branching points $b = l_I + w/2$
    Compute index of a largest edge length $j \in \operatorname{argmax}(\{w_i, i \in [m]\})$
    Compute $c_l = \left|\left\{x_I \in \mathcal{X}_I \;\middle|\; x_{Ij} \leq b_j\right\}\right|, c_g = \left|\left\{x_I \in \mathcal{X}_I \;\middle|\; x_{Ij} \geq b_j\right\}\right|$
    if $c_l < c_g$ **then**
      Update upper bound: $u_{Ij} = \lfloor b_j \rfloor$
      if $c_l < 1$ **then**
        break
      end if
    else
      Update lower bound: $l_{Ij} = \lceil b_j \rceil$
      if $c_g < 1$ **then**
        break
      end if
    end if
    Update set of (relevant) visited integer assignments: $\mathcal{X}_I = \mathcal{X}_I \cap [l_I, u_I]$
  end while
  Return new integer assignment $\hat{x}_I = \lfloor (l_I + u_I)/2 + 0.5e \rfloor$
end procedure

The idea behind this approach is to search for new integer assignments in the "most unexplored" areas of X_I. This is only a heuristic, but works quite well in practice. Moreover, since all objective and constraint functions are continuous it is reasonable to expect that integer assignments that are "close" to each other will also lead to attainable points in roughly the same area. Hence, in order to evenly explore the criterion space it makes sense to search for new integer assignments using this approach.
Nevertheless, this method will need an increasing amount of computation time if the overall number of integer assignments is quite large and $\mathcal{X}_I$ already contains a lot of them. This may imply that a lot of branching steps are needed in order to finally find an empty box (in the sense that it contains no elements of $\mathcal{X}_I$) and hence a new integer assignment.

4.2.3 Fixed boxes

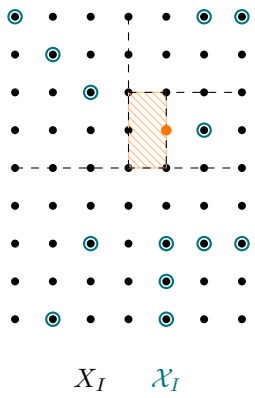
This final approach combines the techniques from the previous two sections. More precisely, it uses a technique to create a predefined number of subboxes of X_I and then computes new integer assignments within those subboxes using full enumeration. As the number and size of boxes is predefined for this approach, we call it the fixed boxes approach.

Algorithm 4 Dividing X_I into a fixed number of subboxes

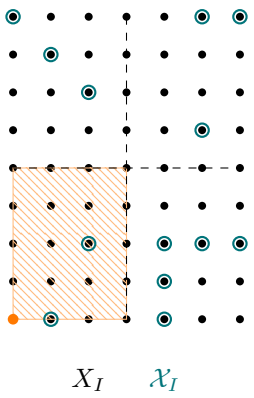
Input: Number of branching steps $b \in \mathbb{N}$, initial box $X_I := [l_I, u_I]$
Output: Set $\mathcal{B}_I$ of subboxes

procedure InitSNIA(b, X_I)
  Compute edge lengths $w = u_I - l_I$
  Initialize $\mathcal{B}_I = \{X_I\}$
  for $i = 1 : b$ **do**
    Compute index of a largest edge length $j \in \operatorname{argmax}(\{w_i, i \in [m]\})$
    if $w_j < 1$ **then**
      break
    end if
    Set $\hat{\mathcal{B}}_I = \emptyset, d = 0_m, d_j = \lceil w_j/2 \rceil$
    for $B = [l, u] \in \mathcal{B}_I$ **do**
      $\hat{\mathcal{B}}_I = \hat{\mathcal{B}}_I \cup \{[l, u - d], [l + d, u]\}$
    end for
    Update $\mathcal{B}_I = \hat{\mathcal{B}}_I, w_j = \lfloor w_j/2 \rfloor$
  end for
end procedure

Let $b \in \mathbb{N}$ be a number of branching steps. Then we compute 2^b subboxes of X_I with equal edge lengths using the maximum edge length as branching criterion, see Algorithm 4. When searching for a new integer assignment (Algorithm 2), we determine a box $B_I \in \mathcal{B}_I$ with minimal $|B_I \cap \mathcal{X}_I|$. Within that box B_I we search for a new integer assignment by full enumeration as described in Section 4.2.1.
This approach is the one that is also used within [22] with $b = 4$. The advantage of this technique is that the set of boxes $\mathcal{B}_I$ has to be computed only once in the beginning of the overall HyPaD algorithm, whereas the dynamic approach in Section 4.2.2 needs to branch and compute new boxes with each call of Algorithm 2. However, in most cases the difference between this approach and the dynamic boxes approach is negligible.
Nonetheless, the different approaches can result in very different integer assignments as shown in Figure 4. In the example from that figure, the dynamic boxes approach performs four branching/update steps as indicated by the dashed black lines in Figure 4 (a). This finally results in the box highlighted in orange which contains non of the previously computed integer assignments $x_I \in \mathcal{X}_I$. The new integer assignment provided by the dynamic boxes approach is the one highlighted in orange.
Figure 4 (b) shows the result of the fixed boxes approach with $b = 2$, i.e., using four boxes, for the exact same scenario. Since the box on the lower left contains the least amount of integer assignments $x_I \in X_I$, this is the box where the new integer assignment is computed. Assuming that the full enumeration within this box starts on the lower left, the integer assignment highlighted in orange is the one that the fixed boxes approach provides as its output.



(a) Dynamic boxes approach



(b) Fixed boxes approach with $b = 2$

Figure 4: Computation of a new integer assignment (highlighted in orange)

5 Numerical experiments

In this section, we present numerical results for 35 test instances, see Table 1. Most test problems are taken from [8], which allows us to compare our algorithm with the algorithms MOMIX and MOMIX light from that paper. The main motivation to compare our algorithm with that from [8] is that, to the best of our knowledge, prior to HyPaD, MOMIX and MOMIX light were the only algorithms that were also able to solve multi-objective mixed-integer convex optimization problems with an arbitrary number of objective functions without using a scalarization-first approach. We want to point out that the instances from [8] are used as test instances in other literature as well, see [4, 10]. We also included two new test problems (T9), (T10) and another new scalable test problem (H1). All of the problem formulations can be found in the appendix at the end of this paper. For a survey and characterization of these and other test instances for multi-objective mixed-integer nonlinear optimization, we refer to [16]. A generator for even more test instances is provided in [15].
All results in this section have been computed using MATLAB R2021a on a machine with Intel Core i9-10920X processor and 32GB of RAM. The average of the results of `bench(5)` is: LU = 0.2045, FFT = 0.2127, ODE = 0.3666, Sparse = 0.3919, 2-D = 0.1968, 3-D = 0.2290. Please be aware that these results of MATLAB's internal benchmark function are version specific, see [32]. All single-objective continuous convex subproblems, in particular (SUP($\hat{x}_I, l, u$)), have been solved using `fmincon`. We also tested other solvers such as IPOPT via the OPTI Toolbox [6], but this did not lead to a significant difference concerning the overall performance of our algorithm. All single-objective mixed-integer linear optimization problems (RSUP($\mathcal{X}, l, u$)) have been solved using Gurobi 9.0.3 [26]. For all instances we set a time limit of 3600 seconds. If this limit was exceeded, we indicate that by a "-" in the tables with the results.
For the initial box $B = [z, Z]$ we provided $\tilde{z}, \tilde{Z} \in \mathbb{R}^p$ as presented in Table 1 and chose $z_i := \tilde{z}_i - 10^{-3}\,\varepsilon, Z_i = \tilde{Z}_i + 10^{-3}\,\varepsilon$ for all $i \in [p]$. Here, $\varepsilon > 0$ denotes the upper bound for the width of the enclosure, which is also provided in Table 1. With the exception of the first subsection, all results for the HyPaD algorithm have been computed using the SNIA procedure using fixed boxes (see Section 4.2.3) with $b = 4$.

number	name	n	m	ε	$\tilde{z}^\top$	$\tilde{Z}^\top$
1	T3	2	1	0.10	$(-2,-2)$	$(2,62)$
2	T3	2	10	0.10	$(-2,-2)$	$(2,80)$
3	T3	2	20	0.10	$(-2,-2)$	$(2,100)$
4	T3	2	30	0.10	$(-2,-2)$	$(2,120)$
5	T4	2	1	0.10	$(-3,-3)$	$(3,3)$
6	T4	2	2	0.10	$(-5,-5)$	$(5,5)$
7	T4	2	3	0.10	$(-7,-7)$	$(7,7)$
8	T4	4	1	0.10	$(-4,-4)$	$(4,4)$
9	T4	2	10	0.10	$(-21,-21)$	$(21,21)$
10	T4	4	10	0.10	$(-22,-22)$	$(22,22)$
11	T4	8	10	0.10	$(-24,-24)$	$(24,24)$
12	T4	2	20	0.10	$(-41,-41)$	$(41,41)$
13	T4	2	20	0.50	$(-41,-41)$	$(41,41)$
14	T4	4	20	0.10	$(-42,-42)$	$(42,42)$
15	T4	2	30	0.10	$(-61,-61)$	$(61,61)$
16	T4	4	30	0.10	$(-62,-62)$	$(62,62)$
17	T4	8	30	0.10	$(-64,-64)$	$(64,64)$
18	T4	16	30	0.10	$(-68,-68)$	$(68,68)$
19	T5	3	1	0.50	$(-3,-3,-1)$	$(3,3,5)$
20	T5	3	1	0.20	$(-3,-3,-1)$	$(3,3,5)$
21	T5	3	1	0.10	$(-3,-3,-1)$	$(3,3,5)$
22	T5	3	1	0.05	$(-3,-3,-1)$	$(3,3,5)$
23	T9	4	4	0.10	$(-3,5)$	$(13,22)$
24	T10	4	4	0.10	$(-3,5)$	$(12,22)$
25	H1	4	10	0.10	$(-14,-14)$	$(34,34)$
26	H1	16	10	0.10	$(-26,-26)$	$(46,46)$
27	H1	64	10	0.10	$(-74,-74)$	$(94,94)$
28	T6	2	1	0.10	$(-3,-1)$	$(3,8.5)$
29	T6	2	1	0.05	$(-3,-1)$	$(3,8.5)$
30	T6	2	1	0.01	$(-3,-1)$	$(3,8.5)$
31	T4	200	2	0.10	$(-14,-14)$	$(14,14)$
32	T4	200	4	0.10	$(-18,-18)$	$(18,18)$
33	T4	200	6	0.10	$(-22,-22)$	$(22,22)$
34	T4	200	8	0.10	$(-26,-26)$	$(26,26)$
35	T4	200	10	0.10	$(-30,-30)$	$(30,30)$

Table 1: List of all test instances including the number $n \in \mathbb{N}$ of continuous and $m \in \mathbb{N}$ of integer variables as well as the choice for the quality parameter $\varepsilon > 0$ and the lower and upper bounds for the initial enclosure

An open source implementation of the HyPaD algorithm is publicly available on GitHub [20]. For the results in this paper we used version 1.0 of the algorithm, which is the same that was used for the numerical results in [22].

5.1 Comparison of fallback procedures for the computation of new integer assignments

First, we compare the results for different realizations of the SNIA procedure, see also Section 4.2. In Table 2 a comparison of the overall computation time, the number of calls of the SNIA procedure, and the computation time needed for the SNIA procedure for all three approaches from Section 4.2 is provided. A more detailed comparison of the dynamic boxes and the fixed boxes approach that also contains the number of calls of the subproblems (RSUP($\mathcal{X}, l, u$)) and (SUP($\hat{x}_I, l, u$)) is shown in Tables 3 and 4.
We observe that for most instances there is almost no difference in terms of overall computation time and the number of calls of SNIA between the full enumeration, the dynamic boxes, and the fixed boxes approach. In fact, even the number of calls of (RSUP($\mathcal{X}, l, u$)) and (SUP($\hat{x}_I, l, u$)) is almost the same in most cases. This holds for the dynamic and fixed boxes approach, see Tables 3 and 4, but also for the full enumeration approach. A possible explanation for this could be that in most cases only a small number of integer assignments is computed by the SNIA procedure (Algorithm 2), i.e., the procedure is called only a few times. In particular, there are only 3 instances where more than 1% of the overall computation time of the HyPaD algorithm is spent on the SNIA procedure. This indicates that in most cases this procedure can really be considered as a fallback for the rarely occurring case that the HyPaD algorithm has no more active integer assignments to work with. Hence, the results match the motivation of the SNIA procedure as presented in [22].
The only instance with noticeable differences between the approaches is instance 17. While the full enumeration and the dynamic boxes approach could solve that instance, this was not the case for the fixed boxes approach. What is more, this is the only instance with a noticeable difference in the number of calls of the SNIA procedure and the overall computation time between the different approaches. In fact, the dynamic boxes approach needs roughly three times as many calls of the SNIA procedure as the full enumeration approach which results in roughly twice the overall computation time. Nevertheless, that instance seems to be a rare exception in our experiments. Consequently, the overall choice for one method over the other should not be based on that one particular result. In fact, we decided to use the fixed boxes approach for all the results in the remaining part of this paper and also in [22]. It is more predictable than the dynamic boxes approach in the sense that we know exactly which and how many boxes are created and it explores the decision space more evenly than the full enumeration approach.

instance	full enumeration			dynamic boxes			fixed boxes ($b = 4$)		
	time	SNIA		time	SNIA		time	SNIA	
		calls	time		calls	time		calls	time
1	3,48	2	0,01	3,80	3	0,03	3,42	2	0,01
2	14,01	0	0,00	13,35	0	0,00	13,33	0	0,00
3	391,31	0	0,00	374,66	0	0,00	373,15	0	0,00
4	-	-	-	-	-	-	-	-	-
5	1,59	1	0,00	1,53	1	0,00	1,48	1	0,00
6	3,57	2	0,03	3,39	2	0,03	3,37	2	0,03
7	5,89	7	0,08	5,47	6	0,07	5,53	7	0,08
8	1,97	1	0,00	1,90	1	0,00	1,87	1	0,00
9	21,51	16	0,16	19,56	12	0,12	19,67	11	0,12
10	31,81	20	0,22	30,92	23	0,25	30,96	23	0,25
11	-	-	-	-	-	-	-	-	-
12	62,31	30	0,29	60,28	30	0,30	60,61	30	0,31
13	10,06	0	0,00	9,46	0	0,00	9,48		
14	105,27	48	0,51	103,51	48	0,52	103,50	48	0,52
15	161,76	60	0,63	160,19	52	0,53	159,89	52	0,59
16	213,62	59	0,68	212,60	58	0,64	213,74	59	0,71
17	304,07	62	0,76	741,70	189	2,18	-	-	-
18	-	-	-	-	-	-	-	-	-
19	1,23	3	0,03	1,16	3	0,03	1,06	3	0,03
20	3,44	3	0,03	3,15	3	0,03	3,20	3	0,03
21	9,22	3	0,03	8,74	3	0,03	8,78	3	0,03
22	28,32	3	0,03	27,00	3	0,03	27,22	3	0,03
23	5,85	0	0,00	5,13	0	0,00	5,11	0	0,00
24	5,08	29	0,49	4,80	28	0,49	4,82	28	0,48
25	133,10	0	0,00	202,93	0	0,00	203,08	0	0,00
26	302,17	0	0,00	407,82	0	0,00	406,87	0	0,00
27	1086,03	0	0,00	1378,83	0	0,00	1373,96	0	0,00
28	1,44	1	0,00	1,34	1	0,00	1,34	1	0,00
29	1,89	1	0,00	1,79	1	0,00	1,80	1	0,00
30	6,23	1	0,00	6,14	1	0,00	6,09	1	0,00
31	173,70	10	1,08	158,10	9	0,87	167,77	9	0,92
32	283,52	10	1,03	277,35	9	0,93	270,55	9	0,93
33	460,61	12	1,30	442,61	13	1,30	416,04	14	1,46
34	612,28	13	1,37	634,94	14	1,39	629,94	14	1,43
35	824,15	14	1,47	855,49	16	1,61	869,63	17	1,74

Table 2: Comparison of overall computation times (in seconds), the number of calls of the SNIA procedure, and time (in seconds) spent on the SNIA procedure for all three realizations of the SNIA procedure

instance	dynamic boxes					fixed boxes ($b = 4$)				
	time	# (RSUP)	# (SUP)	SNIA		time	# (RSUP)	# (SUP)	SNIA	
				calls	time				calls	time
1	3.80	15	15	3	0.03	3.42	15	15	2	0.01
2	13.35	218	15	0	0	13.33	218	15	0	0.00
3	374.66	2698	15	0	0	373.15	2698	15	0	0.00
4	-	-	-	-	-	-	-	-	-	-
5	1.53	30	45	1	0.00	1.48	30	45	1	0.00
6	3.39	70	98	2	0.03	3.37	70	98	2	0.03
7	5.47	111	151	6	0.07	5.53	112	151	7	0.08
8	1.90	30	62	1	0.00	1.87	30	62	1	0.00
9	19.56	334	460	12	0.12	19.67	335	460	11	0.12
10	30.92	387	679	23	0.25	30.96	388	684	23	0.25
11	-	-	-	-	-	-	-	-	-	-
12	60.28	662	905	30	0.30	60.61	664	908	30	0.31
13	9.46	190	81	0	0	9.48	190	81	0	0.00
14	103.51	7727	1350	48	0.52	103.50	776	1355	48	0.52
15	160.19	1039	1404	52	0.53	159.89	1037	1400	52	0.59
16	212.60	1089	1945	58	0.64	213.74	1092	1950	59	0.71
17	741.70	1961	3423	189	2.18	-	-	-	-	-
18	-	-	-	-	-	-	-	-	-	-

Table 3: Comparison of the dynamic boxes and the fixed boxes approach for the SNIA procedure regarding computation times (in seconds) and the number of calls of subproblems and subroutines (part 1)

instance	dynamic boxes					fixed boxes ($b = 4$)				
	time	# (RSUP)	# (SUP)	SNIA		time	# (RSUP)	# (SUP)	SNIA	
				calls	time				calls	time
19	1.16	21	17	3	0.03	1.06	20	15	3	0.03
20	3.15	35	144	3	0.03	3.20	35	149	3	0.03
21	8.74	46	549	3	0.03	8.78	46	549	3	0.03
22	27.00	59	1834	3	0.03	27.22	60	1852	3	0.03
23	5.13	60	108	0	0	5.11	60	108	0	0.00
24	4.80	66	71	28	0.49	4.82	66	71	28	0.48
25	202.93	634	363	0	0	203.08	634	363	0	0.00
26	407.82	868	575	0	0	406.87	868	575	0	0.00
27	1378.83	940	912	0	0	1373.96	940	912	0	0.00
28	1.34	27	36	1	0.00	1.34	27	36	1	0.00
29	1.79	32	64	1	0.00	1.80	32	64	1	0.00
30	6.14	44	379	1	0.00	6.09	44	379	1	0.00
31	158.10	158	594	9	0.87	167.77	156	618	9	0.92
32	277.35	230	900	9	0.93	270.55	228	881	9	0.93
33	442.61	321	1238	13	1.30	416.04	311	1121	14	1.46
34	634.94	407	1523	14	1.39	629.94	405	1530	14	1.43
35	855.49	487	1815	15	1.61	869.63	490	1815	17	1.74

Table 4: Comparison of the dynamic boxes and the fixed boxes approach for the SNIA procedure regarding computation times (in seconds) and the number of calls of subproblems and subroutines (part 2)

5.2 Influence of the choice of ε

In this section, we briefly discuss the effects of the choice of the quality parameter ε. For this, we consider the tri-objective test problem (T5) from [8] that also appears in [4]. We have computed an enclosure of the nondominated set of (T5) for four different choices of $\varepsilon \in \{0.5, 0.2, 0.1, 0.05\}$, see instances 19–22 in Table 1.
Considering the computation time (see Table 5), we notice that halving ε roughly triples the overall computation time. This also holds for the number of calls of `fmincon` within the HyPaD algorithm that make up roughly 80% of the overall computation time. This is not surprising since in this examples there are exactly five integer assignments, which are all feasible integer assignments. All the corresponding patches contribute to the nondominated set and hence, the algorithm basically explores all these integer assignments in the first iterations and computes the overall enclosure of the nondominated set as a combination of the coverages corresponding to the patches. Since this happens almost entirely on the patch level, the calls of `fmincon` make up most of the computation time. This behavior is quite typical for a small number of possible integer assignments.

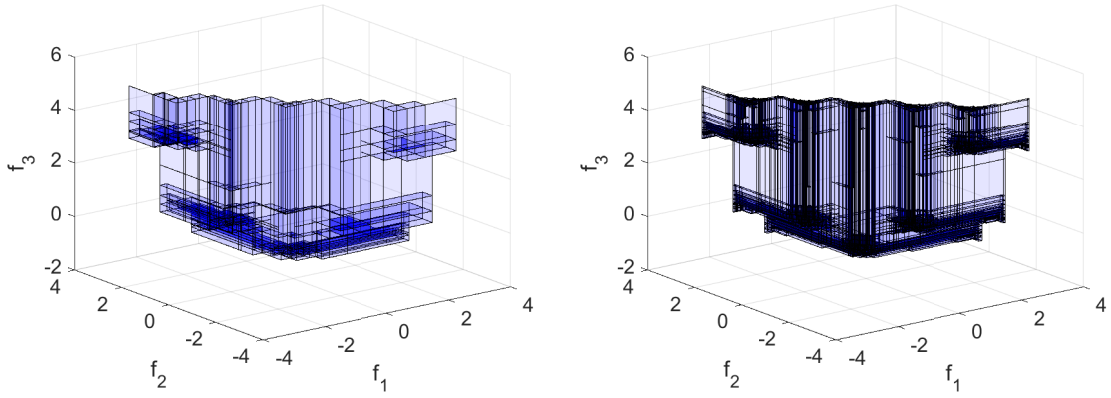
instance	ε	time	$\lvert L\rvert$	$\lvert U\rvert$	`fmincon`	
					calls	time
19	0.50	1.06	35	61	69	0.90
20	0.20	3.20	303	329	218	2.57
21	0.10	8.78	1103	1129	629	7.02
22	0.05	27.22	3709	3735	1946	20.90

Table 5: Results for (T5) with different choices of ε

Besides the overall computation time, another important aspect of the choice of ε is the number of boxes at the end of the algorithm or, more precisely, the number of lower and upper bounds that need to be computed. These numbers are also shown in Table 5. Since the number of lower and upper bounds affects the loop sizes within the HyPaD algorithm and its subroutines, one should be clear that each iteration of the algorithm (i.e., each run of the main while loop) gets more expensive in terms of computation time when the number of lower and upper bounds increases. Hence, there is a noticeable trade-off between the quality of the enclosure and the computation time of the HyPaD algorithm. A visualization of the enclosures for all four choices of ε is given in Figure 5.

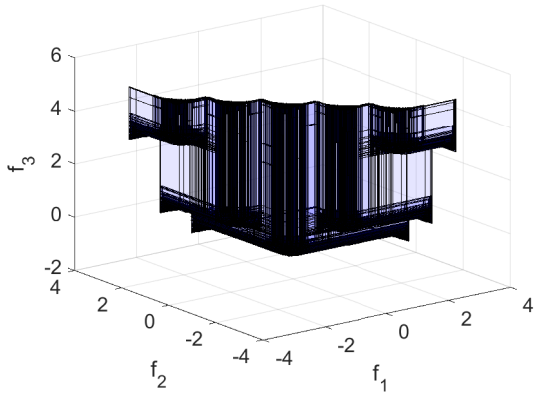
5.3 Number of explored patches

A key ingredient of the HyPaD algorithm is that, for instance compared to [5], it does not assume prior knowledge of the set S_I of feasible integer assignments. In fact, one aim of the algorithm is to compute as few integer assignments as possible. This is also the reason for the constant change between computations for the global lower bound set by solving (RSUP$(\mathcal{X}, l, u)$) and computations for the patch level lower bound sets by solving (SUP$(\hat{x}_I, l, u)$) for some $\hat{x}_I \in S_I$, see also the explanations in Section 3. In the following, we provide numerical evidence that this strategy works out as expected.

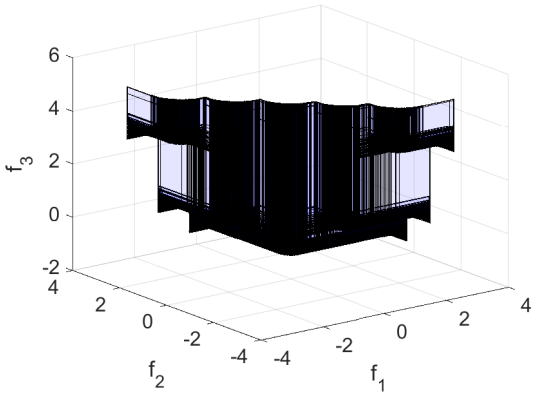


(a) $\varepsilon = 0.5$

(b) $\varepsilon = 0.2$



(c) $\varepsilon = 0.1$



(d) $\varepsilon = 0.05$

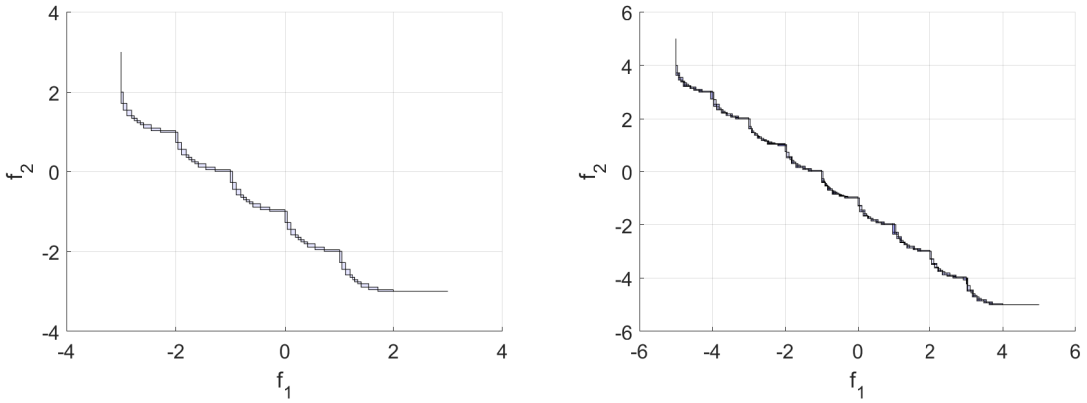
Figure 5: Enclosure for (T5) computed by the HyPaD algorithm for different values of the quality parameter ε

Table 6 shows both the absolute number $P \in \mathbb{N}$ and the share $P/|S_I|$ of feasible integer assignments computed by the HyPaD algorithm. The table contains the results for all test instances with $\varepsilon = 0.1$ that have been solved within the time limit of 3600 seconds and where $X_I = S_I$. As shown in the survey and classification of test instances from [16], the latter is satisfied for all instances of (T4), (T5), (T6), and (H1). Since we only consider instances with $X_I = S_I$, the number P indeed represents the number of all integer assignments computed by the HyPaD algorithm. In particular, a small number of explored patches, i.e., feasible integer assignments computed by the algorithm, cannot arise due to other influences such as a large number of infeasible integer assignments that are computed first.
There are three key observations regarding the results from Table 6. The first observation is that while an increasing number $|S_I|$ can also result in an increasing number P of explored patches, the share $P/|S_I|$ usually decreases. The best example for this are instances 31 to 35 where the share decreases from 76.00% for $|S_I| = 5^2$ to less than 0.01% for $|S_I| = 5^{10}$. In fact, the share for instance 35 is only $6.04 \cdot 10^{-6}$.

instance	problem	n	m	$\lvert S_I \rvert$	flag	time	P	$P/\lvert S_I \rvert$
5	T4	2	1	5	0	1.48	5	100.00%
6	T4	2	2	25	1	3.37	13	52.00%
7	T4	2	3	125	1	5.53	21	16.80%
8	T4	4	1	5	0	1.87	5	100.00%
9	T4	2	10	5^{10}	1	19.67	59	$< 0.01\%$
10	T4	4	10	5^{10}	1	30.96	65	$< 0.01\%$
12	T4	2	20	5^{20}	1	60.61	117	$< 0.01\%$
14	T4	4	20	5^{20}	1	103.50	130	$< 0.01\%$
15	T4	2	30	5^{30}	1	159.89	184	$< 0.01\%$
16	T4	4	30	5^{30}	1	213.74	182	$< 0.01\%$
21	T5	3	1	5	0	8.78	5	100.00%
25	H1	4	10	5^{10}	1	203.08	399	$< 0.01\%$
26	H1	16	10	5^{10}	1	406.87	509	0.01%
27	H1	64	10	5^{10}	1	1373.96	492	0.01%
28	T6	2	1	5	0	1.34	5	100.00%
31	T4	200	2	25	1	167.77	19	76.00%
32	T4	200	4	625	1	270.55	28	4.48%
33	T4	200	6	5^6	1	416.04	40	0.26%
34	T4	200	8	5^8	1	629.94	48	0.01%
35	T4	200	10	5^{10}	1	869.63	59	$< 0.01\%$

Table 6: Absolute number P and share $P/\lvert S_I \rvert$ of feasible integer assignments computed by the HyPaD algorithm, overall computation time (in seconds) and exitflag

The second observation is that for all instances with only five possible integer assignments, i.e., instances 5, 8, 21, and 28, all of these integer assignments have been computed by the HyPaD algorithm. What is more, for these instances the lower bound set for the enclosure of the overall nondominated set of (MOMIP) has indeed been computed as a combination of the patch level lower bound sets. This is indicated in Table 6 by the value of the algorithm's exitflag. It is 1 in case the HyPaD algorithm terminates using the global lower bound set obtained by solving (RSUP($\mathcal{X}, l, u$)), see also line 8 of Algorithm 5, and 0 in case it terminates using the patch level lower bound sets for the enclosure, see line 5 of Algorithm 2. Hence, for this particular kind of optimization problems the HyPaD algorithm reduces the computation of the enclosure for the nondominated set of the multi-objective mixed-integer convex optimization problem (MOMIP) to mostly solving single-objective continuous convex optimization problems (SUP($\hat{x}_I, l, u$)).

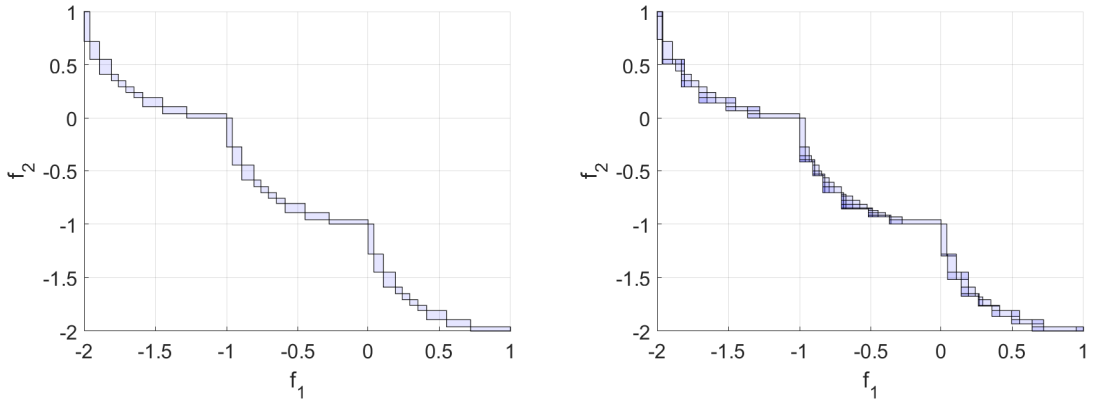


(a) Results for $m = 1$ (test instance 5) (b) Results for $m = 2$ (test instance 6)

Figure 6: Comparison of the enclosure computed by the HyPaD algorithm for (T4) with $n = 2$ continuous and $m \in \{1, 2\}$ integer variables

In Figure 6 we provide a visual comparison of the enclosures computed by the HyPaD algorithm for test instances 5 and 6. Both test instances correspond to (T4) with $n = 2$ continuous variables. While test instance 5 has only a single integer variable ($m = 1$) and hence only five possible integer assignments, test instance 6 has $m = 2$ integer variables and $|X_I| = |S_I| = 25$ possible integer assignments. In particular, for test instance 5 the HyPaD algorithm terminates with exitflag 0 and computes the final lower bound set using the patch level lower bound sets $L_{\hat{x}_I}$. For test instance 6 it terminates with exitflag 1 and uses the global lower bound set L, which is updated using weakly efficient solutions of (R($\mathcal{X}$)). This also results in structural differences of the enclosures as shown in Figure 7. For test instance 5 the coverages of the nondominated sets of the patch problems are very homogeneous. This is mainly because all of the patch problems possess a similar structure and their nondominated sets have no overlap. Hence, the patch level lower bound sets are almost identical for all patch problems despite an offset based on the corresponding integer assignments. In Figure 7 (b), one clearly sees that for test instance 6 the lower bound set is computed globally. In particular, there is not the same kind of similarity of the lower bounds with regard to the nondominated sets of the different patch problems.

The special structure of the lower bound set for test instance 5 is related to the so-called separable structure (also separability) of (T4). For more details on that, we refer to [15]. An even broader classification of the properties of test instances for multi-objective mixed-integer nonlinear optimization is provided in [16].



(a) Results for test instance 5 ($m = 1$)

(b) Results for test instance 6 ($m = 2$)

Figure 7: Comparison of the enclosure structure for test instances 5 and 6 (visualization of the criterion space restricted to $[-2, 1]^2$)

The third observation with regard to the results from Table 6 is that, in general, only a small number of feasible integer assignments needs to be computed by the HyPaD algorithm. For most instances from Table 6, especially those with a large number of feasible integer assignments, only 0.01% or even less of them had to be computed by the HyPaD algorithm in order to compute an enclosure of the nondominated set of prescribed quality. This also means that the global lower bounding strategy successfully avoids the need of a full enumeration of all feasible integer assignments in order to compute an enclosure. This was exactly the motivation to introduce this strategy in [22], see also Section 3.

5.4 Comparison of HyPaD and MOMIX

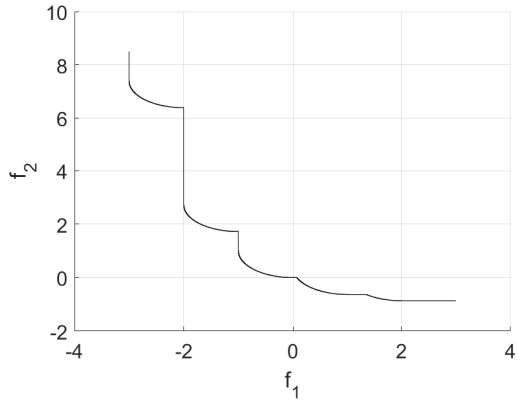
In this final section, we compare the HyPaD algorithm with the MOMIX algorithm from [8]. To make this comparison as fair as possible, we decided not to use the results from [8], but to compute our own results using the MOMIX code provided on GitHub [7]. Hence, both algorithms are using the same machine, the same solvers for the subproblems and so on.

Before we present the actual results, we recap briefly the different properties and operation modes of MOMIX. The MOMIX algorithm includes a procedure to obtain tight lower bounds by solving a single-objective mixed-integer convex optimization problem that is basically of the same type as the original problem (MOMIP). For example, if the original problem is quadratically constrained, this holds for the subproblems as well. Most MIP solvers (e.g., Gurobi [26] or CPLEX [29]) can solve such optimization problems as long as the objective and constraint functions are at least quadratic. Since not all convex functions are quadratic functions, there is also a weaker version of MOMIX, called MOMIX light, that uses another procedure for the computation of lower bounds. That procedure is based on solving a single-objective continuous convex optimization problem (using `fmincon`).

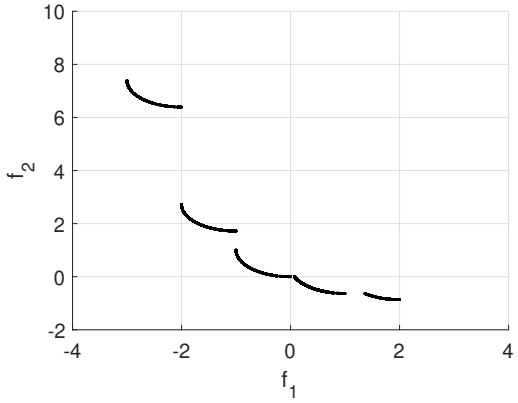
Both variants of the algorithm can use two different branching strategies, see [8, Section 4.1]. The strategy (br1) is an integer first branching strategy. The second strategy (br2) uses the largest edge length as branching criterion, even if it is related to a continuous variable. In total this leads to four different operation modes of the MOMIX algorithm and we present the results for all of them.
One difference between MOMIX and the HyPaD algorithm is that MOMIX is a branch-and-bound approach in the decision space while the HyPaD algorithm operates almost entirely in the criterion space. Since MOMIX works in the decision space, it also computes a coverage of the set of efficient solutions. This is not the case for the HyPaD algorithm. This also leads to a difference with respect to the quality criteria. While we use the width of the enclosure $w(\mathcal{C})$, which is a criterion space based measure, MOMIX uses the box width in the decision space as termination criterion. This box width is limited by an input parameter $\delta > 0$ in the same way that the width of the enclosure computed by the HyPaD algorithm is limited by the parameter $\varepsilon > 0$. At least for MOMIX (not MOMIX light) there is a result related to the width concept of the enclosure $\mathcal{C}$, see [8, Theorem 3.13]. For most instances we set $\varepsilon = \delta = 0.1$ and for instances 19–22 we fixed $\delta = 0.5$ and varied the parameter ε to demonstrate that for the overall qualitative comparison of the two algorithms this has no major impact.
Another difference between HyPaD and MOMIX is that the single-objective mixed-integer subproblem (RSUP$(\mathcal{X}, l, u)$) within the HyPaD algorithm is always linear, independently of the type of objective and constraint functions of (MOMIP). As a result, it can always use standard MILP solvers (e.g., Gurobi [26] or CPLEX [29]) to solve these subproblems, even if one of the objective or constraint functions is non-quadratic. Since we include the results for both MOMIX and MOMIX light, we leave it to the reader to decide which of them would make for the "fairest" comparison to the HyPaD algorithm. All results are shown in Table 7.
First of all, MOMIX light has either high computation times or exceeds the time limit of 3600 seconds for most of the instances. Thus, when MOMIX is not an option (e.g., in case of non-quadratic objective or constraint functions), one should definitely consider using HyPaD over MOMIX light. One perfect example for this setting is the test problem (T6) (instances 28–30) which has a non-quadratic objective function. Even if there are only $n = 2$ continuous variables and a single integer variable ($m = 1$), MOMIX light needs more than 1000 seconds to solve the corresponding instance with $\delta = 0.1$. HyPaD on the other hand profits from the small number of possible integer assignments and computes an enclosure in less than 10 seconds even for $\varepsilon = 0.01$. For a visual comparison of the results see Figure 8.
Also MOMIX has its strengths when it can be applied, i.e., when all objective and constraint functions are quadratic. For the instances of problem (T3) with 20 and 30 integer variables, i.e., instances 3 and 4, MOMIX with (br2) clearly outperforms HyPaD. However, this is only one of two (considering only MOMIX and not MOMIX light) possible configurations of MOMIX and it is an open question whether there is a method to detect beforehand that this is the right configuration to choose. In particular, MOMIX with (br2) is not always the best option, see for example instance 21. A visualization of the results for test instance 3 is given in Figure 9.

instance	HyPaD	MOMIX light		MOMIX	
		(br1)	(br2)	(br1)	(br2)
1	3.42	637.06	629.49	8.04	7.85
2	13.33	-	-	13.86	13.53
3	373.15	-	-	369.19	26.12
4	-	-	-	-	45.19
5	1.48	171.48	128.03	22.65	24.41
6	3.37	2497.56	1980.76	129.13	135.79
7	5.53	-	-	752.27	773.47
8	1.87	-	-	1334.60	1318.00
9	19.67	-	-	-	-
10	30.96	-	-	-	-
11	-	-	-	-	-
12	60.61	-	-	-	-
14	103.50	-	-	-	-
15	159.89	-	-	-	-
16	213.74	-	-	-	-
17	-	-	-	-	-
18	-	-	-	-	-
21	8.78	-	-	89.72	105.58
22	27.22	-	-	89.72	105.58
28	1.34	-	1385.72		
29	1.80	-	1385.72		
30	6.09	-	1385.72		
31	167.77	-	-	-	-
32	270.55	-	-	-	-
33	416.04	-	-	-	-
34	629.94	-	-	-	-
35	869.63	-	-	-	-

Table 7: Comparison of computation times (in seconds) for HyPaD, MOMIX, and MOMIX light

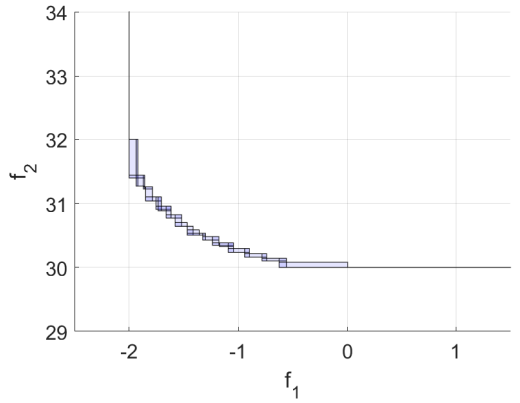


(a) Enclosure computed by HyPaD

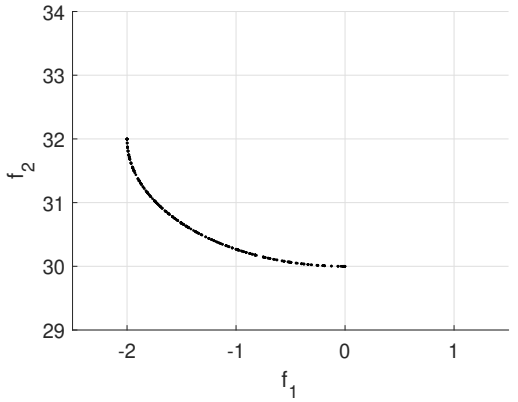


(b) Representation computed by MOMIX

Figure 8: Results for test instance 30



(a) Enclosure computed by HyPaD

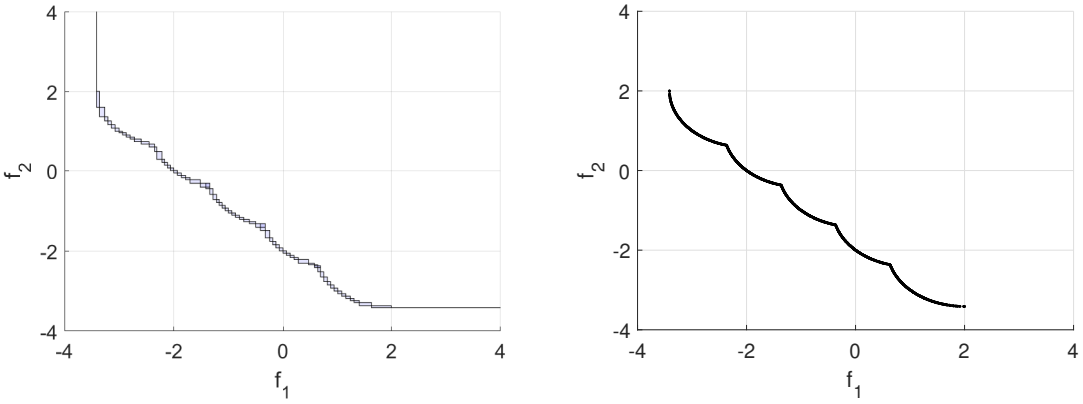


(b) Representation computed by MOMIX

Figure 9: Results for test instance 3

Regarding the instances of problem (T4), i.e., instances 5–18 and 31–35, HyPaD is performing better than MOMIX and is also able to solve more of the given instances within the specified time limit of 3600 seconds. For a comparison of the results, see the visualization of test instance 8 in Figure 10. Compared to instances 3 and 4 where MOMIX was able to handle a large number of integer variables, this seems not to be the case for problem (T4), see instances 9–18.
We also used (T4) to test the performance of HyPaD on instances with a large number of variables. For this reason, we chose a large number of continuous variables ($n = 200$) and varied the number of integer variables $m \in \{2, 4, 6, 8, 10\}$, see instances 31–35. We also tested instances with $m \in \{10, 20, 30\}$ integer variables and a smaller number of continuous variables, see instances 9–18. First of all, we realize that all instances with $n = 200$ continuous variables were solved within the time limit of 3600 seconds. For $m \in \{10, 20, 30\}$ the algorithm successfully solved those instances with $n < 8$ continuous variables, but for $n = 8$ it exceeded the time limit, see instances 11 and 17. In particular, the HyPaD algorithm was able to solve instance 35 with $n = 200$ and $m = 10$ within 869.63 seconds, but exceeded the time limit for instance 11 with

$n = 8$ and $m = 10$. A possible explanation for this could be that the nondominated set of instance 11 has a more complex (more non-linear) shape than the one of instance 35. Thus the nondominated set of instance 35 is probably better approximated by the nondominated set of the corresponding linearized problems ($\mathrm{R}(\mathcal{X})$), which leads to a faster convergence of the global lower bound set L towards the upper bound set U in that setting.



(a) Enclosure computed by HyPaD

(b) Representation computed by MOMIX

Figure 10: Results for test instance 8

Also for problem (T5) in instances 21 and 22 HyPaD is clearly ahead of MOMIX. The HyPaD algorithm is able to solve the problem for $\varepsilon = 0.1$ within only 9 seconds and even for $\varepsilon = 0.05$ this only increases to 27 seconds which is roughly a third of the best performance obtained by MOMIX with (br1).

Acknowledgements

This work was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - Project-ID 432218631.

References

[1] N. Adelgren and A. Gupte, *Branch-and-bound for biobjective mixed-integer linear programming*, INFORMS Journal on Computing, 34 (2022), pp. 909–933.

[2] N. Boland, H. Charkhgard, and M. Savelsbergh, *A criterion space search algorithm for biobjective mixed integer programming: The triangle splitting method*, INFORMS Journal on Computing, 27 (2015), pp. 597–618.

[3] P. Bonami, L. T. Biegler, A. R. Conn, G. Cornuéjols, I. E. Grossmann, C. D. Laird, J. Lee, A. Lodi, F. Margot, N. Sawaya, and A. Wächter, *An algorithmic framework for convex mixed integer nonlinear programs*, Discrete Optimization, 5 (2008), pp. 186–204.

[4] R. S. Burachik, C. Y. Kaya, and M. M. Rizvi, *Algorithms for generating pareto fronts of multi-objective integer and mixed-integer programming problems*, Engineering Optimization, 54 (2021), pp. 1413–1425.

[5] G. Cabrera-Guerrero, M. Ehrgott, A. J. Mason, and A. Raith, *Bi-objective optimisation over a set of convex sub-problems*, Annals of Operations Research, 319 (2022), pp. 1507–1532.

[6] J. Currie, *OPTI toolbox*. https://github.com/jonathancurrie/OPTI. Accessed 2023-02-23.

[7] M. De Santis, G. Eichfelder, J. Niebling, and S. Rocktäschel, *MOMIX*. https://github.com/mariannadesantis/MOMIX. Accessed 2023-02-23.

[8] ——, *Solving multiobjective mixed integer convex optimization problems*, SIAM Journal on Optimization, 30 (2020), pp. 3122–3145.

[9] M. De Santis, G. Eichfelder, D. Patria, and L. Warnow, *Using dual relaxations in multiobjective mixed-integer quadratic programming*, Preprint 23303, Optimization Online, 2023.

[10] E. Diessel, *An adaptive patch approximation algorithm for bicriteria convex mixed-integer problems*, Optimization, 71 (2022), pp. 4321–4366.

[11] M. A. Duran and I. E. Grossmann, *An outer-approximation algorithm for a class of mixed-integer nonlinear programs*, Mathematical Programming, 36 (1986), pp. 307–339.

[12] K. Dächert and K. Teichert, *An improved hyperboxing algorithm for calculating a Pareto front representation*, Preprint 2003.14249, arXiv, 2020.

[13] M. Ehrgott and X. Gandibleux, *Bound sets for biobjective combinatorial optimization problems*, Computers & Operations Research, 34 (2007), pp. 2674–2694.

[14] G. Eichfelder, *Twenty years of continuous multiobjective optimization in the twenty-first century*, EURO Journal on Computational Optimization, 9 (2021). 100014.

[15] G. Eichfelder, T. Gerlach, and L. Warnow, *A test instance generator for multiobjective mixed-integer optimization*, Mathematical Methods of Operations Research, (2023). DOI: 10.1007/s00186-023-00826-z.

[16] ——, *Test instances for multiobjective mixed-integer nonlinear optimization*, Preprint 22458, Optimization Online, 2023.

[17] G. Eichfelder, P. Kirst, L. Meng, and O. Stein, *A general branch-and-bound framework for continuous global multiobjective optimization*, Journal of Global Optimization, 80 (2021), pp. 195–227.

[18] G. Eichfelder, O. Stein, and L. Warnow, *A solver for multiobjective mixed-integer convex and nonconvex optimization*, Journal of Optimization Theory and Applications, (2023). DOI: 10.1007/s10957-023-02285-2.

[19] G. Eichfelder and L. Warnow, *An approximation algorithm for multi-objective optimization problems using a box-coverage*, Journal of Global Optimization, 83 (2022), pp. 329–357.

[20] ——, *HyPaD*. https://github.com/LeoWarnow/HyPaD, 2022. Accessed 2023-02-23.

[21] ——, *Advancements in the computation of enclosures for multi-objective optimization problems*, European Journal of Operational Research, 310 (2023), pp. 315–327.

[22] ——, *A hybrid patch decomposition approach to compute an enclosure for multi-objective mixed-integer convex optimization problems*, Mathematical Methods of Operations Research, (2023). DOI: 10.1007/s00186-023-00828-x.

[23] S. Esmaeili, M. Bashiri, and A. Amiri, *An exact criterion space search algorithm for a bi-objective blood collection problem*, European Journal of Operational Research, 311 (2023), pp. 210–232.

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

[25] R. Fletcher and S. Leyffer, *Solving mixed integer nonlinear programs by outer approximation*, Mathematical Programming, 66 (1994), pp. 327–349.

[26] Gurobi Optimization LLC, *Gurobi*. https://www.gurobi.com/. Accessed 2023-04-17.

[27] A. Göpfert, H. Riahi, C. Tammer, and C. Zalinescu, *Variational Methods in Partially Ordered Spaces*, Springer, 2003.

[28] P. Halffmann, L. E. Schäfer, K. Dächert, K. Klamroth, and S. Ruzika, *Exact algorithms for multiobjective linear optimization problems with integer variables: A state of the art survey*, Journal of Multi-Criteria Decision Analysis, 29 (2022), pp. 341–363.

[29] IBM, *CPLEX optimizer*. https://www.ibm.com/analytics/cplex-optimizer. Accessed 2023-04-17.

[30] K. Klamroth, R. Lacour, and D. Vanderpooten, *On the representation of the search region in multi-objective optimization*, European Journal of Operational Research, 245 (2015), pp. 767–778.

[31] M. Link and S. Volkwein, *Adaptive piecewise linear relaxations for enclosure computations for nonconvex multiobjective mixed-integer quadratically constrained programs*, Journal of Global Optimization, 87 (2023), pp. 97–132.

[32] MATLAB, *Matlab bench documentation*. https://www.mathworks.com/help/matlab/ref/bench.html. Accessed 2023-04-17.

[33] G. Mavrotas and D. C. Diakoulaki, *A branch and bound algorithm for mixed zero-one multiple objective linear programming*, European Journal of Operational Research, 107 (1998), pp. 530–541.

[34] ——, *Multi-criteria branch and bound: A vector maximization algorithm for mixed 0-1 multiple objective linear programming*, Applied Mathematics and Computation, 171 (2005), pp. 53–71.

[35] L. Paquete, B. Schulze, M. Stiglmayr, and A. C. Lourenço, *Computing representations using hypervolume scalarizations*, Computers & Operations Research, 137 (2022). 105349.

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

[37] T. Perini, N. Boland, D. Pecin, and M. Savelsbergh, *A criterion space method for biobjective mixed integer programming: The boxed line method*, INFORMS Journal on Computing, 32 (2020), pp. 16–39.

[38] A. Przybylski, K. Klamroth, and R. Lacour, *A simple and efficient dichotomic search algorithm for multi-objective mixed integer linear programs*, Preprint 1911.08937, arXiv, 2019.

[39] S. A. B. Rasmi and M. Türkay, *GoNDEF: an exact method to generate all non-dominated points of multi-objective mixed-integer linear programs*, Optimization and Engineering, 20 (2019), pp. 89–117.

[40] R. Roozbahani, B. Abbasi, and S. Schreider, *Optimal allocation of water to competing stakeholders in a shared watershed*, Annals of Operations Research, 229 (2015), pp. 657–676.

[41] S. R. Rump, *INTLAB - INTerval LABoratory*, in Developments in Reliable Computing, T. Csendes, ed., Kluwer Academic Publishers, Dordrecht, 1999, pp. 77–104.

[42] S. Ruzika and M. M. Wiecek, *Approximation methods in multiobjective programming*, Journal of Optimization Theory and Applications, 126 (2005), pp. 473–501.

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

[44] S. K. Singh and M. Goh, *Multi-objective mixed integer programming and an application in a pharmaceutical supply chain*, International Journal of Production Research, 57 (2018), pp. 1214–1237.

[45] B. Soylu and G. B. Yildiz, *An exact algorithm for biobjective mixed integer linear programming problems*, Computers & Operations Research, 72 (2016), pp. 204–213.

[46] T. Stidsen and K. A. Andersen, *A hybrid approach for biobjective optimization*, Discrete Optimization, 28 (2018), pp. 89–114.

[47] A.-J. Ulusoy, F. Pecci, and I. Stoianov, *Bi-objective design-for-control of water distribution networks with global bounds*, Optimization and Engineering, 23 (2021), pp. 527–577.

[48] P. Xidonas, G. Mavrotas, and J. Psarras, *Equity portfolio construction and selection using multiobjective mathematical programming*, Journal of Global Optimization, 47 (2010), pp. 185–209.

[49] J. Xu and W. Yang, *Multi-objective steel plate cutting optimization problem based on real number coding genetic algorithm*, Scientific Reports, 12 (2022). 22472.

[50] K. Yang, M. Emmerich, A. Deutz, and T. Bäck, *Efficient computation of expected hypervolume improvement using box decomposition algorithms*, Journal of Global Optimization, 75 (2019), pp. 3–34.

[51] Özgür Özpeynirci and M. Köksalan, *An exact algorithm for finding extreme supported nondominated points of multiobjective mixed integer programs*, Management Science, 56 (2010), pp. 2302–2315.

Appendix

Test problems

The following four test problems are taken from [8]. The first one is a bi-objective optimization problem with quadratic objective and constraint functions. It has $n = 2$ continuous variables and an arbitrary number $m \in \mathbb{N}$ of integer variables.

$$\min \left(x_1, x_2 + \sum_{i=3}^{2+m} 10(x_i - 0.4)^2 \right)^\top \quad \text{s.t.} \quad \begin{aligned} &\sum_{i=1}^{2+m} x_i^2 \leq 4, \\ &x_C \in [-2, 2]^2, \\ &x_I \in [-2, 2]^m \cap \mathbb{Z}^m \end{aligned} \tag{T3}$$

The next test problem is also bi-objective. It has linear objective functions but a quadratic constraint function. The number $m \in \mathbb{N}$ of integer variables can be chosen arbitrarily. The number $n \in \mathbb{N}$ of continuous variables has to be even.

$$\min \left(\sum_{i=1}^{n/2} x_i + \sum_{i=n+1}^{n+m} x_i, \sum_{i=n/2+1}^{n} x_i - \sum_{i=n+1}^{n+m} x_i \right)^\top \quad \text{s.t.} \quad \begin{aligned} &\sum_{i=1}^{n} x_i^2 \leq 1, \\ &x_C \in [-2, 2]^n, \\ &x_I \in [-2, 2]^m \cap \mathbb{Z}^m \end{aligned} \tag{T4}$$

The following tri-objective problem has a fixed number of $n = 3$ continuous and $m = 1$ integer variables. It has a quadratic objective functions as well as a quadratic constraint function.

$$\min \left(x_1 + x_4, x_2 - x_4, x_3 + x_4^2 \right)^\top \quad \text{s.t.} \quad \begin{aligned} &\sum_{i=1}^{3} x_i^2 \leq 1, \\ &x_C \in [-2, 2]^3, \\ &x_I \in [-2, 2] \cap \mathbb{Z} \end{aligned} \tag{T5}$$

The last test problem from [8] is also quadratically constrained, but has a non-quadratic objective function. Both, the number $n = 2$ of continuous and the number $m = 1$ of integer variables are fixed.

$$\begin{aligned} \min\ & (x_1 + x_3, x_2 + \exp(-x_3))^\top \quad \text{s.t.} \quad x_1^2 + x_2^2 \leq 1, \\ & \qquad\qquad\qquad\quad\ \ x_C \in [-2, 2]^2, \\ & \qquad\qquad\qquad\quad\ \ x_I \in [-2, 2] \cap \mathbb{Z} \end{aligned} \tag{T6}$$

The next two test problems are new and have been created to investigate the influence of a quadratic over a non-quadratic objective function. Both problems have quadratic constraint functions and a fixed number of $n = 4$ continuous and $m = 4$ integer variables. The first problem is the one with purely linear (and thus quadratic) objective functions.

$$\begin{aligned} \min\ & \begin{pmatrix} x_1 + x_3 + x_5 + x_7 \\ x_2 + x_4 + x_6 + x_8 \end{pmatrix} \quad \text{s.t.} \quad x_1^2 + x_2^2 \leq 1, \\ & \qquad\qquad\qquad\qquad\quad\ \ x_3^2 + x_4^2 \leq 1, \\ & \qquad\qquad\qquad\qquad\quad\ \ (x_5 - 2)^2 + (x_6 - 5)^2 \leq 10, \\ & \qquad\qquad\qquad\qquad\quad\ \ (x_7 - 3)^2 + (x_8 - 8)^2 \leq 10, \\ & \qquad\qquad\qquad\qquad\quad\ \ x_C \in [-20, 20]^4, \\ & \qquad\qquad\qquad\qquad\quad\ \ x_I \in [-20, 20]^4 \cap \mathbb{Z}^4 \end{aligned} \tag{T9}$$

Test problem (T10) is basically the same as (T9) just with a slightly changed first objective function that is now non-quadratic.

$$\begin{aligned} \min\ & \begin{pmatrix} x_1 + x_3 + x_5 + \exp(x_7) - 1 \\ x_2 + x_4 + x_6 + x_8 \end{pmatrix} \quad \text{s.t.} \quad x_1^2 + x_2^2 \leq 1, \\ & \qquad\qquad\qquad\qquad\quad\ \ x_3^2 + x_4^2 \leq 1, \\ & \qquad\qquad\qquad\qquad\quad\ \ (x_5 - 2)^2 + (x_6 - 5)^2 \leq 10, \\ & \qquad\qquad\qquad\qquad\quad\ \ (x_7 - 3)^2 + (x_8 - 8)^2 \leq 10, \\ & \qquad\qquad\qquad\qquad\quad\ \ x_C \in [-20, 20]^4, \\ & \qquad\qquad\qquad\qquad\quad\ \ x_I \in [-20, 20]^4 \cap \mathbb{Z}^4 \end{aligned} \tag{T10}$$

We want to mention that for both problems (T9) and (T10) the boxes X_C and X_I could have been chosen smaller. However, we wanted the HyPaD algorithm to compute the final approximation using the global lower bound set L that is computed using the optimal solutions of the mixed-integer linear optimization problems (RSUP($\mathcal{X}, l, u$)). This would not have happened if there was only a small number of possible integer assignments (i.e., a small box X_I).

The last test problem is a new scalable one with quadratic objective functions and a quadratic constraint function. Both the number $n \in \mathbb{N}$ of continuous variables and the number $m \in \mathbb{N}$ of integer variables have to be even.

$$\min \begin{pmatrix} \sum_{i=1}^{n/2} x_i + \sum_{i=n+1}^{n+m/2} x_i^2 - \sum_{i=n+m/2+1}^{n+m} x_i \\ \sum_{i=n/2+1}^{n} x_i - \sum_{i=n+1}^{n+m/2} x_i + \sum_{i=n+m/2+1}^{n+m} x_i^2 \end{pmatrix} \quad \text{s.t.} \quad \begin{aligned} &\sum_{i=1}^{n} x_i^2 \leq 1, \\ &x_C \in [-2,2]^n, \\ &x_I \in [-2,2]^m \cap \mathbb{Z}^m \end{aligned} \tag{H1}$$

For a more extensive survey on test instances for multi-objective mixed-integer optimization and a characterization and classification of all the test instances used in this paper, we refer to [16]. We also remark that a generator for new test instances is provided in [15].

HyPaD Algorithm

In this section, we include the pseudocode for the HyPaD algorithm and its main subroutines from [22]. As a prerequisite we need to introduce the so-called feasibility problem

$$\min_{x_C, \alpha} \ \alpha \quad \text{s.t.} \quad \begin{aligned} &g_j(x_C, \hat{x}_I) \leq \alpha \quad \forall j \in [q], \\ &x_C \in X_C, \ \alpha \in \mathbb{R} \end{aligned} \tag{F($\hat{x}_I$)}$$

where $\hat{x}_I \in X_I$ is an arbitrary integer assignment. This procedure allows to detect whether an integer assignment obtained by an optimal solution of (RSUP($\mathcal{X}, l, u$)) is feasible (and hence corresponds to a patch (P($\hat{x}_I$))) or infeasible (and hence can be excluded from future iterations).

The main procedure of HyPaD is shown in Algorithm 5. The improvement step, which also includes the SNIA procedure, is presented in Algorithm 6. For completeness, we also included Algorithms 1 and 7 that are used for initializing and updating the coverages of the patches. However, that part of the algorithm is not discussed in this paper. For more details, we refer the reader to [22]. Finally, we remark that, compared to the presentation in [22], we slightly generalized the initialization of Algorithm 5 and adjusted Algorithm 7, since we incorporated the generalized local upper and local lower bound concepts from [21], see also Section 2.2.

Algorithm 5 Hybrid patch decomposition algorithm for (MOMIP)

Input: Initial point $\hat{x} \in X$, quality parameter $\varepsilon > 0$, and initial enclosure $A = \mathcal{C}(L', U')$ satisfying Assumption 2.4

Output: Lower and upper bound sets $L, U \subseteq \mathbb{R}^p$

procedure HyPaD$(\hat{x}, \varepsilon, z, Z)$
  Initialize $L = L'$, $U = U'$, $\mathcal{X} = \{\hat{x}\}$, $\mathcal{D} = ()$
  Solve (F($\hat{x}_I$)) with optimal solution $(\bar{x}_C, \bar{\alpha})$ and set $\bar{x} := (\bar{x}_C, \hat{x}_I)$
  if $\bar{\alpha} \leq 0$ **then**
    InitIDS($\hat{x}_I$) ▷ see Algorithm 1
    Update linearization points: $\mathcal{X} = \mathcal{X} \cup \mathcal{D}.E$
  end if
  while $w(\mathcal{C}(L, U)) > \varepsilon$ **do**
    for $l \in L$ **do**
      if $(\{l + \varepsilon e\} + \text{int}(\mathbb{R}^p_+)) \cap U \neq \emptyset$ **then**
        Select $u \in (\{l + \varepsilon e\} + \text{int}(\mathbb{R}^p_+)) \cap U$ with maximal $s(l, u)$
        Solve (RSUP($\mathcal{X}, l, u$)) with optimal solution $(\bar{x}, \bar{\eta}, \bar{t})$
        Update lower bound set: $L =$ UpdateLLB$(L, \bar{\eta})$
        Solve (F($\bar{x}_I$)) with optimal solution $(\hat{x}_C, \hat{\alpha})$ and set $\hat{x} := (\hat{x}_C, \bar{x}_I)$
        if $\hat{\alpha} \leq 0$ **then**
          Improve$(\hat{x}, \varepsilon, U, \mathcal{X}, \mathcal{D})$ ▷ see Algorithm 6
          Update linearization points: $\mathcal{X} = \mathcal{X} \cup \mathcal{D}.E$
        else
          Update linearization points: $\mathcal{X} = \mathcal{X} \cup \{\hat{x}\}$
        end if
      end if
    end for
  end while
end procedure

Algorithm 6 Improvement Step on Patch-Level

Input: Feasible point $\hat{x} \in S$, quality $\varepsilon > 0$, upper bound set U, set of linearization points $\mathcal{X}$, and integer data structure $\mathcal{D}$

Output: Updated sets $U, \mathcal{X}$, and updated data structure $\mathcal{D}$

procedure Improve$(\hat{x}, \varepsilon, U, \mathcal{X}, \mathcal{D})$
  if $\mathcal{D}(\hat{x}_I)$ is not initialized **then**
    InitIDS($\hat{x}_I$) ▷ see Algorithm 1
  else if $\mathcal{D}(\hat{x}_I).S ==$ true **then**
    UpdateIDS$(\hat{x}_I, \varepsilon, U, \mathcal{D})$ ▷ see Algorithm 7
  else if $\exists\, x'_I \in S_I$ with $\mathcal{D}(x'_I)$ initialized and $\mathcal{D}(x'_I).S ==$ true **then**
    UpdateIDS$(x'_I, \varepsilon, U, \mathcal{D})$ ▷ see Algorithm 7
  else
    SNIA$(\mathcal{X}, \mathcal{D})$ ▷ see Algorithm 2
  end if
end procedure

Algorithm 7 Updating $\mathcal{D}(\hat{x}_I)$ for an integer assignment $\hat{x}_I \in S_I$

Input: Integer assignment $\hat{x}_I \in S_I$, quality $\varepsilon > 0$, upper bound set U, and data structure $\mathcal{D}$

Output: Updated set U and updated integer data structure $\mathcal{D}$

procedure UPDATEIDS($\hat{x}_I, \varepsilon, U, \mathcal{D}$)
  Initialize $L_{\hat{x}_I} = \mathcal{D}(\hat{x}_I).L$, done = true
  for $l \in L_{\hat{x}_I}$ **do**
    if $(\{l + \varepsilon e\} + \text{int}(\mathbb{R}^p_+)) \cap U \neq \emptyset$ **then**
      done = false
      Select $u \in (\{l + \varepsilon e\} + \text{int}(\mathbb{R}^p_+)) \cap U$ with maximal $s(l, u)$
      Solve (SUP($\hat{x}_I, l, u$)) with optimal solution $(\bar{x}_C, \bar{t})$ and set $\bar{x} := (\bar{x}_C, \hat{x}_I)$, $\bar{y} := f(\bar{x})$ and $\tilde{y} := l + \bar{t}(u - l)$
      $\mathcal{D}(\hat{x}_I).E = \mathcal{D}(\hat{x}_I).E \cup \{\bar{x}\}$
      $\mathcal{D}(\hat{x}_I).N = \mathcal{D}(\hat{x}_I).N \cup \{\bar{y}\}$
      $\mathcal{D}(\hat{x}_I).L = \text{UPDATELLB}(\mathcal{D}(\hat{x}_I).L, \tilde{y})$
      $U = \text{UPDATELUB}(U, \bar{y})$
    end if
  end for
  if done == true **then**
    Set integer assignment inactive: $\mathcal{D}(\hat{x}_I).S = \text{false}$
  end if
end procedure