On implementation details and numerical experiments for the HyPaD algorithm to solve multi-objective mixed-integer convex optimization problems

Gabriele Eichfelder*, Leo Warnow*

August 22, 2023

Abstract

In this paper, we present insights on the implementation details of the hybrid patch decomposition algorithm (HyPaD) for multi-objective mixed-integer convex optimization problems. We discuss three methods to implement the SNIA procedure which is basically a black box algorithm in the original work by Eichfelder and Warnow. In addition, we present and discuss numerical results for various test instances. We also give some advice on how to choose the parameter ε for the width of the computed enclosure of the nondominated set.

Key Words: multi-objective optimization, mixed-integer optimization, numerical experiments, enclosure

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

1 Introduction

Multi-objective optimization problems that have some continuous and also some integer variables are denoted multi-objective mixed-integer optimization problems. In our work we consider the optimization problem

$$\min_x \ f(x) \quad \text{s.t.} \quad g(x) \leq 0_q, \ x \in X := X_C \times X_I \tag{MOMICP}$$

for once continuously differentiable convex objective functions $f_i \colon \mathbb{R}^{n+m} \to \mathbb{R}$, $i \in [p]$ and once continuously differentiable convex constraint functions $g_j \colon \mathbb{R}^{n+m} \to \mathbb{R}$, $j \in [q]$ where $[p] := \{1, \ldots, p\}$, $f = (f_1, \ldots, f_p) \colon \mathbb{R}^{n+m} \to \mathbb{R}^p$, $g = (g_1, \ldots, g_q) \colon \mathbb{R}^{n+m} \to \mathbb{R}^q$ and $0_q \in \mathbb{R}^q$ denotes the all-zeros vector. We assume that $X_C := [l_C, u_C] \subseteq \mathbb{R}^n$ is a nonempty box with $l_C, u_C \in \mathbb{R}^n$ and $X_I := [l_I, u_I] \cap \mathbb{Z}^m$ is 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 the optimization problem (MOMICP).

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

The feasible set of (MOMICP) is denoted by S and is assumed to be nonempty. Moreover, we define the set of all feasible integer assignments by

$$S_I = \{x_I \in \mathbb{Z}^m \mid \exists\, x_C \in \mathbb{R}^n \colon (x_C, x_I) \in S\}$$

and call every $x_I \in S_I$ a feasible integer assignment. For every $\hat{x}_I \in S_I$ we denote by $S_{\hat{x}_I} := \{x \in S \mid x_I = \hat{x}_I\}$ the subset of feasible points for (MOMICP) with exactly that assignment of the integer variables.

In [9] we presented the Hybrid Patch Decomposition algorithm (HyPaD) to compute an enclosure of the nondominated set of (MOMICP). We make use of the enclosure concept as presented in [7], but restricted to finite sets $L, U \subseteq \mathbb{R}^p$.

Definition 1.1 *Let $L, U \subseteq \mathbb{R}^p$ be two finite sets such that $\mathcal{N} \subseteq L + \mathbb{R}^p_+$ and $\mathcal{N} \subseteq U - \mathbb{R}^p_+$. Then L is called a lower bound set, U is called an upper bound set, and the set $\mathcal{A}$ which is given as*

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

is called the enclosure of the nondominated set $\mathcal{N}$ of (MOMICP) given L and U.

The HyPaD algorithm is guaranteed to compute such an enclosure with a width of at most $\varepsilon > 0$, where ε is an input parameter of the algorithm. The width of an enclosure $\mathcal{A}$ is denoted by $w(\mathcal{A})$ and equals the optimal value of

$$\max_{l,u} \; s(l, u) \quad \text{s.t.} \quad l \in L,\; u \in U,\; l \leq u \tag{1.1}$$

where $s(l, u) := \min\{u_i - l_i \mid i \in [p]\}$ denotes the shortest edge length of a box $[l, u]$. This concept is also presented in [7]. Using the width as a quality criterion, one can assure that any $x \in S$ with $f(x) \in \mathcal{A}(L, U)$ is an ε-efficient solution of (MOMICP) if $w(\mathcal{A}) < \varepsilon$ for some $\varepsilon > 0$, see also [7, Lemma 3.1].

The main contribution of this paper is to extend the primarily theoretical results from [9] in mainly two directions. First, we provide details on the theoretical and practical realization of the HyPaD algorithm regarding steps that have originally been considered as a black box. This includes the initialization of the algorithm in Section 2 and, more importantly, the realization of the SNIA procedure in Section 3 of this paper. In fact, we present three methods to realize this fallback procedure which is used within the HyPaD algorithm to compute new integer assignments $x_I \in X_I$ in order to guarantee progress and thus finiteness of the algorithm even in a worst case scenario.

The second half of this paper, namely Section 4, then focuses on numerical experiments to evaluate the strengths and weaknesses of the HyPaD algorithm. In Section 4.1, we discuss the influence of the different realizations for the SNIA procedure presented in Section 3 with regard to the overall performance of the algorithm. We further investigate the influence of the parameter ε that determines the width, i.e., the quality of the computed enclosure, in Section 4.2. Finally, in Section 4.3, we compare the HyPaD algorithm from [9] to the MOMIX algorithm from [3]. The latter was the first and prior to the introduction of HyPaD the only algorithm in the literature that also specifically addressed multi-objective mixed-integer convex optimization problems with an arbitrary number of objective functions and that did not rely on a scalarization-first approach.

We remark that in the remaining part of this paper we use many of the concepts and notations from [9]. We briefly recall those concepts, like the SNIA procedure, that are relevant for the discussions and results presented in this paper in the corresponding sections. We also include the pseudocode of the algorithm and all its subroutines in the appendix of this paper (starting from page 21). For everything beyond that we refer the reader to [9].

2 Initialization

For the initialization of the HyPaD algorithm (Algorithm 5), we need a starting point $\hat{x} \in X$ and an initial box $B := [z, Z]$ with $f(S) \subseteq \operatorname{int}(B)$.

In this paper, especially with regard to the numerical experiments in Section 4, we always provide the initial box B manually and do not compute it within MATLAB. However, if one wants to include a mechanism to compute an initial box within MATLAB, then using interval arithmetic (for example via INTLAB [13]) would be a suitable approach to do that. In fact, both ways for the initialization of B are included in our implementation of the HyPaD algorithm which we provide on GitHub [8].

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 (MOMICP):

$$\begin{aligned} \min_{x,t} \; t \quad \text{s.t.} \quad & f(x) - z - t(Z - z) \leq 0_p, \\ & g(x) \leq 0_q, \\ & x \in X_C \times [l_I, u_I] \subseteq \mathbb{R}^{n+m}, \; t \in \mathbb{R}. \end{aligned} \tag{$\mathrm{P}_{\mathrm{init}}$}$$

We denote by $(\bar{x}, \bar{t})$ an optimal solution of ($\mathrm{P}_{\mathrm{init}}$). 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., $\hat{x} = (\bar{x}_C, \lfloor \bar{x}_I + 0.5e \rfloor) \in X$, where $e \in \mathbb{R}^m$ denotes the all-ones vector.

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. Let $\hat{x}_I \in S_I$ be such a new feasible integer assignment. We define the corresponding patch (problem) as

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

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 ($\mathrm{P}(\hat{x}_I)$). 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 ($\mathrm{P}(\hat{x}_I)$), 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 ($\mathrm{P}(\hat{x}_I)$) 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 (MOMICP). 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 ($\mathrm{P}(\hat{x}_I)$) 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 ($\mathrm{P}(\hat{x}_I)$) that are computed within the algorithm.

In [9] the initialization of $\mathcal{D}(\hat{x}_I)$ is almost treated as a black box and it is not further specified how the first lower bound $\hat{z} \in \mathbb{R}^p$ with $f(S_{\hat{x}_I}) \subseteq \{\hat{z}\} + \operatorname{int}(\mathbb{R}^p_+)$ for the corresponding patch is computed. 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

$$\min_{x_C} \; f_i(x_C, \hat{x}_I) \quad \text{s.t.} \quad g(x_C, \hat{x}_I) \leq 0_q, \; x_C \in X_C. \tag{PI($\hat{x}_I, i$)}$$

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 the ideal point for the patch corresponding to the integer assignment $\hat{x}_I \in S_I$. By using a small offset $\sigma > 0$ (e.g., $\sigma = 10^{-3}\,\varepsilon$) this allows us to initialize the integer data structure $\mathcal{D}(\hat{x}_I)$ as shown in Algorithm 1.

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

procedure InitIDS($\hat{x}_I$)
  For all $i \in [p]$ solve (PI($\hat{x}_I, i$)) with optimal solution $\bar{x}^i_C$ and optimal value $\bar{z}_i \in \mathbb{R}$
  Compute $\hat{z} \in \mathbb{R}^p$ as $\hat{z}_i := \bar{z}_i - \sigma$ for all $i \in [p]$
  Initialize $\mathcal{D}(\hat{x}_I).L = \{\hat{z}\}$, $\mathcal{D}(\hat{x}_I).E = \{(\bar{x}^1_C, \hat{x}_I), \ldots, (\bar{x}^p_C, \hat{x}_I)\}$, $\mathcal{D}(\hat{x}_I).N = \{f(\bar{x}^1_C, \hat{x}_I), \ldots, f(\bar{x}^p_C, \hat{x}_I)\}$, $\mathcal{D}(\hat{x}_I).S = \text{true}$
end procedure

3 Realization of the SNIA procedure

The HyPaD algorithm is basically an interplay of computing integer assignments $x_I \in X_I$ by solving the single-objective mixed-integer linear optimization problem

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

with $l, u \in \mathbb{R}^p, l < u$, and a finite nonempty set $\mathcal{X} \subseteq \mathbb{R}^{n+m}$, and improving the coverages of the nondominated sets $\mathcal{N}_{\hat{x}_I} \subseteq f(S)$ of patches which belong to feasible integer assignments $\hat{x}_I \in S_I$. Since all coverages on the patch level need to be improved only finitely often, see [9, Lemma 6.4], it can happen that all integer assignments that have been computed by HyPaD so far are either infeasible or inactive, i.e., the corresponding coverages need no further improvement. For the correctness of the overall algorithm, one needs to ensure that HyPaD will not get stuck in this situation. This can be achieved by forcing HyPaD 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 computed by HyPaD, Algorithm 2 computes such a new integer assignment. More precisely, this is done in

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**

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 [9].

Concerning the theory presented in [9], it is not important how the new integer assignment is computed, see 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 $k := |X_I|$.

3.1 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 in 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 3.2 and 3.3. In particular, this makes full enumeration a very good choice for two scenarios. The first one are problems (MOMICP) 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 (MOMICP) 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.

3.2 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 image 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.

3.3 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.

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 the 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 3.1.

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

This approach is the one that we actually use within [9] where we set $b = 4$. 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 3.2 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.

4 Numerical experiments

In this section, we present numerical results for 35 test instances, see Table 1. Most test problems are taken from [3], which allows us to compare our algorithm with the algorithms `MOMIX` and $\texttt{MOMIX}_{\texttt{light}}$ from that paper. The main motivation to compare our algorithm with that from [3] is that, to the best of our knowledge, prior to HyPaD, `MOMIX` and $\texttt{MOMIX}_{\texttt{light}}$ 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 [3] are used as test instances in other literature as well, see [1, 5]. 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 [6].

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 benchmarking function are version specific, see [12]. 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 [2], 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 [11]. 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 $i \in [p]$. With the exception of the first subsection, all results for HyPaD have been computed using the SNIA procedure using fixed boxes (see Section 3.3) with $b = 4$.

The MATLAB implementation of the HyPaD algorithm is publicly available on GitHub [8]. The raw data for the numerical experiments is provided on Zenodo [10].

4.1 Comparison of SNIA approaches

First, we compare the results for different realizations of the SNIA procedure, see also Section 3. 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 3 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).

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 computed instances

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, calls of the SNIA procedure, and time 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 (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 (part 2)

This means that 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 [9].

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 [9] since it is more predictable than the dynamic boxes approach in the sense that we know exactly what and how many boxes are created and since it explores the decision space more evenly than the full enumeration approach.

4.2 Influence of the choice of ε

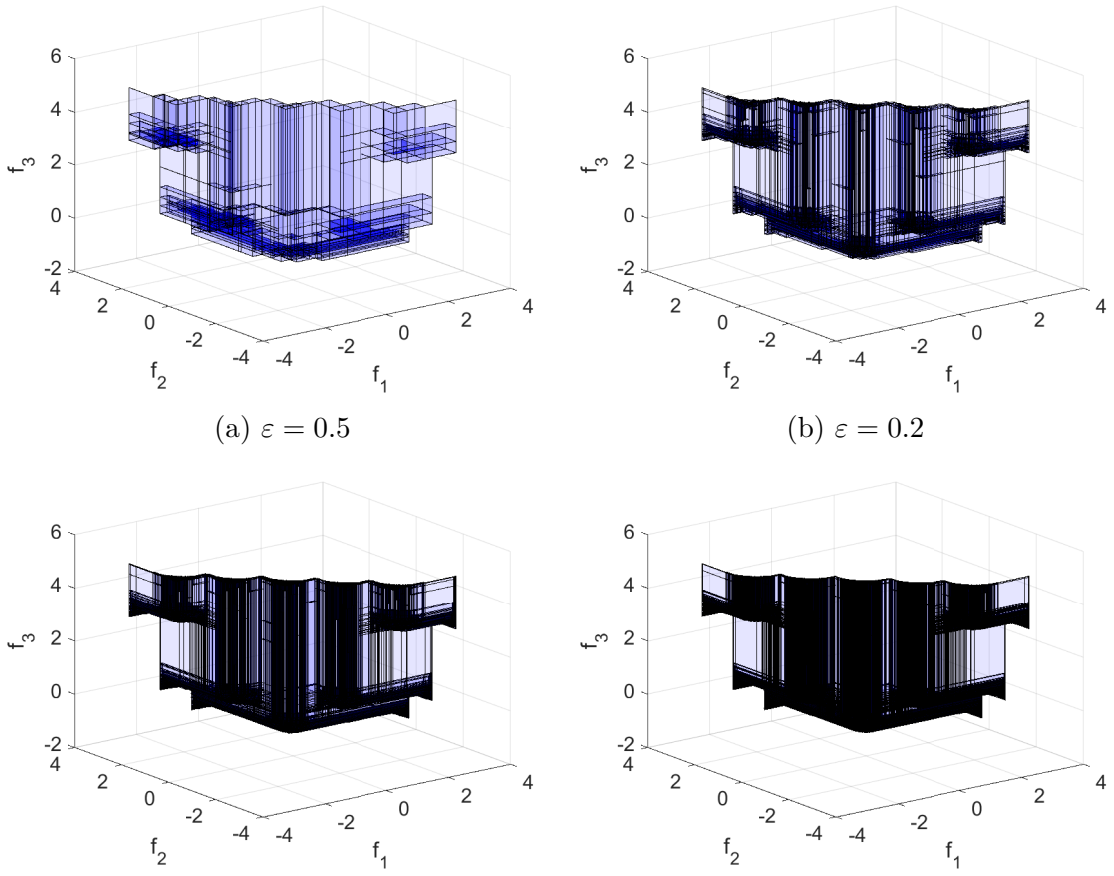
In this section, we briefly discuss the effects of the choice of the quality parameter ε. For this, we consider the tri-objective test problem (T5) from [3] that also appears in [1]. We have computed an enclosure of the nondominated set of (T5) for four different choices of $\varepsilon \in \{0.5, 0.2, 0.1, 0.05\}$, see instances 19–22 in Table 1.

Considering 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 HyPaD 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, HyPaD 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`	time `fmincon`
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 HyPaD and its subroutines, one should be clear that each iteration of the algorithm (i.e., each run of the main while loop of HyPaD) 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 HyPaD.



(a) $\varepsilon = 0.5$ (b) $\varepsilon = 0.2$

(c) $\varepsilon = 0.1$ (d) $\varepsilon = 0.05$

Figure 1: Enclosure for (T5) computed by HyPaD for different values of ε

4.3 Comparison of HyPaD and MOMIX

In this final section, we compare our HyPaD algorithm with the MOMIX algorithm from [3]. To make this comparison as fair as possible, we decided not to use the results from [3], but to compute our own results using the MOMIX code provided on GitHub [4]. Hence, both algorithms are using the same machine, the same versions of Gurobi [11] 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 (MOMICP).

For example, if the original problem is quadratically constrained, this holds for the subproblems as well. Such optimization problems can be solved by Gurobi 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 [3, 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 our algorithm HyPad is that MOMIX is a branch-and-bound approach in the decision space while our algorithm is working 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 $w(\mathcal{A})$, which is a criterion space based measure, MOMIX uses the box width in the decision space as termination criterion. At least for MOMIX (not MOMIX light) there is a result related to our width concept of the enclosure $\mathcal{A}$, see [3, Theorem 3.13]. However, for most instances we set $\varepsilon = \delta = 0.1$ and for instances 19–22 we fixed $\delta = 0.5$ and varied our 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 our single-objective mixed-integer subproblem (RSUP$(\mathcal{X}, l, u)$) is always linear and in that sense independent of the type of objective and constraint functions of (MOMICP). As a result, we can always use Gurobi 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 our algorithm. All results are shown in Table 6.

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 2.

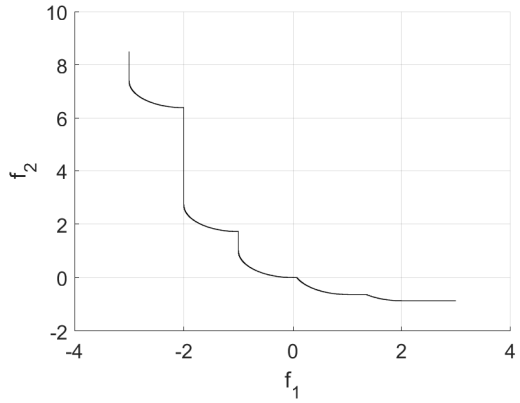
Also MOMIX has its strengths when it can be applied, i.e., when all objective and constraint functions are quadratic. For the instances of problem (T3) with 20 and 30 integer variables, i.e., instances 3 and 4, MOMIX with (br2) clearly outperforms HyPaD. However, this is only one of two (considering 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 3.

instance	HyPaD	MOMIX$_{\texttt{light}}$ (br1)	MOMIX$_{\texttt{light}}$ (br2)	MOMIX (br1)	MOMIX (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	-	-	-	-

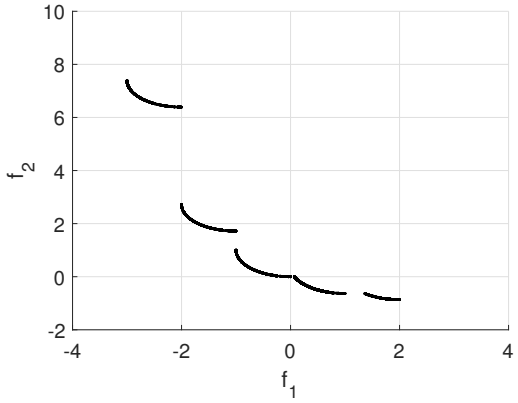
Table 6: Comparison of computation times for HyPaD, MOMIX, and MOMIX$_{\texttt{light}}$

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 4. 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. (The reason for fixing $n = 200$ is that for even larger instances there are numerical issues when calling `fmincon`.) 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

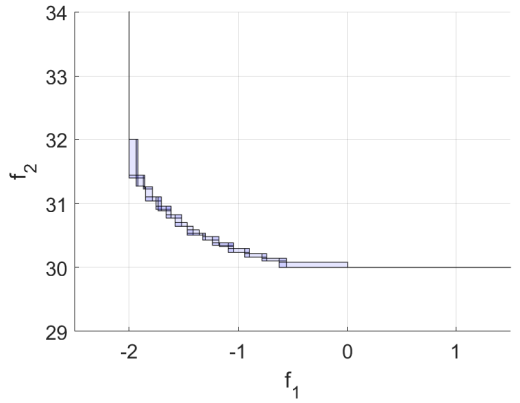


(a) Enclosure computed by HyPaD

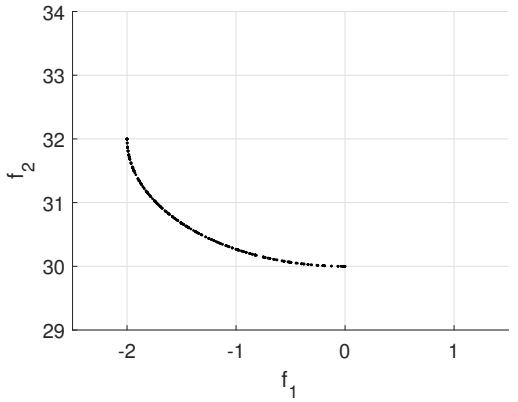


(b) Representation computed by MOMIX

Figure 2: Results for test instance 30



(a) Enclosure computed by HyPaD

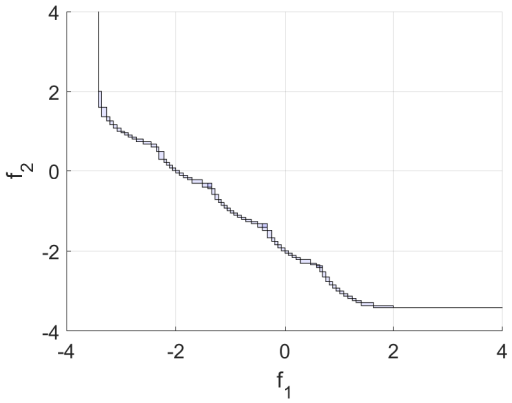


(b) Representation computed by MOMIX

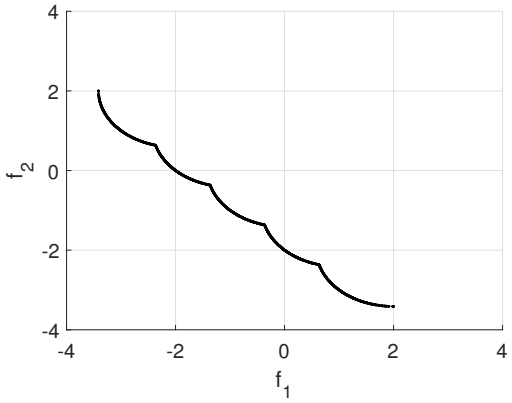
Figure 3: Results for test instance 3

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.

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).



(a) Enclosure computed by HyPaD



(b) Representation computed by MOMIX

Figure 4: Results for test instance 8

5 Acknowledgements

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

References

[1] 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 (2022), pp. 1413–1425.

[2] J. Currie, *OPTI toolbox*. https://github.com/jonathancurrie/OPTI, 2019. Accessed 2023-02-23.

[3] M. De Santis, G. Eichfelder, J. Niebling, and S. Rocktäschel, *Solving multiobjective mixed integer convex optimization problems*, SIAM Journal on Optimization, 30 (2020), pp. 3122–3145.

[4] M. De Santis, G. Eichfelder, J. Niebling, and S. Rocktäschel, *MOMIX*. https://github.com/mariannadesantis/MOMIX, 2021. Accessed 2023-02-23.

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

[6] G. Eichfelder, T. Gerlach, and L. Warnow, *Test Instances for Multiobjective Mixed-Integer Nonlinear Optimization*, Preprint 22458, Optimization Online, 2023.

[7] 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.

[8] G. Eichfelder and L. Warnow, *HyPaD*. https://github.com/LeoWarnow/HyPaD, 2022. Accessed 2023-02-23.

[9] G. Eichfelder and L. Warnow, *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.

[10] G. Eichfelder and L. Warnow, *Results of the numerical experiments of the HyPaD algorithm*. Zenodo, 2023. DOI: 10.5281/zenodo.8119013.

[11] Gurobi Optimization LLC, *Gurobi*. https://www.gurobi.com/, 2023. Accessed 2023-02-23.

[12] MATLAB, *Matlab bench documentation*. https://www.mathworks.com/help/matlab/ref/bench.html, 2023. Accessed 2023-02-23.

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

Appendix

Test problems

The following four test problems are taken from [3]. The first one is a bi-objective optimization problem with quadratic objective and constraint functions. It has $n = 2$ continous 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 [3] 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.

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

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.

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

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

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

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 our algorithm HyPaD 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 that we introduced in [9]. Both the number $n \in \mathbb{N}$ of continuous variables and the number $m \in \mathbb{N}$ of integer variables have to be even.

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

HyPaD Algorithm

In this section we include the pseudocode for our HyPaD algorithm and its main subroutines from [9]. As a prerequisite we need the procedure to search for an update point. For $\hat{x}_I \in S_I$ and $l, u \in \mathbb{R}^p$ with $l < u$, this is done by solving

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

Further, we need the feasibility problem

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

where $\hat{x}_I \in X_I$ is an integer assignment.

Algorithm 5 Hybrid patch decomposition algorithm for (MOMICP)

Input: Initial point $\hat{x} \in X$, quality $\varepsilon > 0$, and initial bounds $z, Z \in \mathbb{R}^p$
Output: Lower and upper bound sets $L, U \subseteq \mathbb{R}^p$

procedure HyPaD($\hat{x}, \varepsilon, z, Z$)
  Initalize $L = \{z\}$, $U = \{Z\}$, $\mathcal{X} = \{\hat{x}\}$, $\mathcal{D} = ()$
  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 $w(\mathcal{A}(L, U)) > \varepsilon$ **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: $L = \text{UpdateLLB}(L, \bar{\eta})$
        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

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 Algorithm 1 and Algorithm 7 that are used for initializing and updating the coverages of the single patches. However, that part of the algorithm is not discussed in this paper and hence, for more details on that we refer the reader to the original paper [9].

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
  Choose a small offset $\sigma \in (0, \varepsilon/2)$
  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 $\tilde{y} := l + \bar{t}(u - l)$
      if $\tilde{y} \not\leq Z$ **then**
        Set $\bar{t} := \min\{(Z_i - l_i)/(u_i - l_i) \mid i \in [p]\}$ and $\tilde{y} := l + \bar{t}(u - l) - \sigma e$
      end if
      $\mathcal{D}(\hat{x}_I).E = \mathcal{D}(\hat{x}_I).E \cup \{\bar{x}\}$
      $\mathcal{D}(\hat{x}_I).N = \mathcal{D}(\hat{x}_I).N \cup \{\bar{y}\}$
      $\mathcal{D}(\hat{x}_I).L =$ UPDATELLB($\mathcal{D}(\hat{x}_I).L, \tilde{y}$)
      $U =$ UPDATELUB($U, \bar{y}$)
    end if
  end for
  if done == true **then**
    Set integer assignment inactive: $\mathcal{D}(\hat{x}_I).S =$ false
  end if
end procedure