Mathematical Programming Computation
https://doi.org/10.1007/s12532-026-00308-3

FULL LENGTH PAPER

libDIPS – discretization-based semi-infinite and bilevel programming solvers

Daniel Jungen[2] · Aron Zingler[2] · Hatim Djelassi[2] · Alexander Mitsos[1,2,3]

Received: 13 November 2024 / Accepted: 11 February 2026

Abstract
We consider several hierarchical optimization problems: (generalized) semi-infinite and existence-constrained semi-infinite programs, minmax, and bilevel programs. Multiple adaptive discretization-based algorithms have been published for these problem classes in recent decades. However, rigorous numerical performance comparisons between these algorithms are lacking. Indeed, if numerical comparisons are provided at all, they typically compare a small selection of algorithms on small test sets, across different platforms, and with various subsolvers required during the solution. Additionally, some algorithms have parameters that impede a fair comparison. Our contribution is threefold: i) We present an open-source software called libDIPS (Discretization-based semi-Infinite and bilevel Programming Solvers), which implements multiple adaptive discretization-based solvers. The primary benefit of libDIPS is that it enables users to flexibly switch between the implemented solvers within a single problem class and switch between the available subsolvers. ii) We compile an extensive library of test problems for the (generalized) semi-infinite, minmax, and bilevel problem classes, which, in total, contains over 600 problem instances. Our set includes eight merged test sets and additional problem instances from over 80 literature sources. iii) We compare the solvers numerically using our library of test problems and identify tradeoffs in the parameter tuning.

Mathematics Subject Classification 90-04 · 90C34

1 Introduction

Generalized semi-infinite and semi-infinite programs ((G)SIPs), existence-constrained semi-infinite programs (ESIPs), minmax programs (MINMAX), and bilevel programs

✉ Alexander Mitsos
amitsos@alum.mit.edu

1 JARA-CSD, Aachen 52056, Germany

2 Process Systems Engineering (AVT.SVT), RWTH Aachen University, Aachen 52074, Germany

3 Institute of Climate and Energy Systems: Energy Systems Engineering (ICE-1), Forschungszentrum Jülich GmbH, Jülich 52425, Germany

Published online: 11 May 2026

(BLPs) are hierarchical optimization problems that occur in many applications, e.g., gemstone cutting [141] and thermodynamics [22, 47], or when considering worst-case uncertainties as in robust optimization [10, 31, 43] and flexibility analysis [125]. These problems belong to the class of hierarchical optimization problems because the definition of their objective or constraints involves solving a lower-level optimization problem. In this manuscript, we focus on deterministic hierarchical optimization problems, excluding probabilistic problem formulations. A reliable and fast solution to such optimization problems is paramount but challenging: the lower-level optimization problem must be solved globally, even to determine the feasibility of a given candidate solution point. If the lower-level problems of (G)SIPs, MINMAX, or BLPs are convex and satisfy certain regularity conditions, the hierarchical optimization problem can be reformulated as a single-level problem. However, in many applications, the lower-level problems are nonconvex.

Multiple theoretical approaches have been developed over the past few decades for the global solution of hierarchical optimization poblems without convexity assumptions, with substantial recent advances. These include classical discretization methods, adaptive discretization-based approaches, and adaptations thereof, overestimation methods using interval methods and relaxation methods, optimal value function approaches, and relaxation-based branch-and-bound methods [13–15, 29, 34, 38, 81, 83, 89, 92, 111, 117, 131]. See [31] for more details.

Here, we focus on adaptive discretization-based approaches. A key benefit of these adaptive discretization-based approaches is that they can utilize well-established optimization solvers, primarily as a black box for solving subproblems. Moreover, (adaptive) discretization-based algorithms are relatively general: they support integer variables at both levels [28, 88] and, apart from any specific requirements imposed by the subsolvers, require mainly compact domains and continuous functions. Their practicality is further demonstrated by successful applications in both research and industry [27, 73]. In recent decades, numerous adaptive discretization-based algorithms have been published to solve different problem classes, as shown in Table 2.

Evaluating the relative performance of these algorithms is difficult because computational performance is usually assessed on different platforms and implementations. In addition, many algorithms use inherent algorithmic tuning parameters (in the following simply called parameters) that complicate direct comparison due to their substantial influence on performance. Finally, no unified set of test problems is consistently used across publications, so performance evaluation is typically based on an individual and small set of test problems. To remedy this, we implement multiple adaptive discretization-based algorithms within a coherent software framework and compile an extensive library of test problems for comparison.

While we focus on comparing the implemented adaptive discretization-based algorithms, these algorithms are already used as a point of comparison in numerical evaluations of other algorithms. Specifically, [20, 70, 83] compared the numerical performance of their approaches against the algorithms proposed by [29, 89, 90]. For their comparisons, [20] used their own implementations of [89], [83] used the GAMS implementation provided by [29], and [70] used the data reported by [90]. The additional effort needed to reimplement the respective adaptive discretization-based algorithm, especially without access to the commercially distributed subsolvers

or the ability to conduct comparisons based solely on the original publication, may have deterred researchers from using it as a reference point. An alternative to reimplementation would be to use one of the publicly available solvers for hierarchical optimization problems. Table 1 provides an overview, focusing on those solvers that support nonlinearity and nonconvexity at one of the levels. However, potential reasons preventing their use in a comparison are: limitations placed on the problem type at the upper or lower level, the lack of a deterministic global solution, or that the relevant problem class, i.e., (G)SIP or BLP, is not supported. Additionally, some solvers are commercially distributed or utilize proprietary dependencies.

We remedy this by implementing multiple adaptive discretization-based algorithms in a coherent open-source software called libDIPS – Discretization-based semi-Infinite and bilevel Programming Solvers. libDIPS provides solvers for the deterministic global solution of (G)SIPs, ESIPs, MINMAXes, and BLPs with upper- and lower-levels of MINLP type. Our software is not only freely available and user-friendly, but it also serves as an ideal framework for implementing both existing and new algorithms, particularly making the implementation of discretization-based algorithms straightforward. Additionally, we compile an extensive library of test problems to compare the algorithms. The combination of libDIPS and the library of test problems arguably makes further comparisons between algorithms easier for other researchers and makes the solution of hierarchical optimization problems more accessible. Finally, we compare the solvers numerically using our library of test problems and identify tradeoffs in the parameter tuning. In summary, we provide (i) a software framework for the solution of hierarchical optimization problems, (ii) compile a library of problem instances, and use the former two to (iii) run comparative tests of adaptive discretization-based algorithms to identify tradeoffs in the parameter tuning.

In Sect. 2, we briefly review the formulation of the problem classes (G)SIP, ESIP, MINMAX, and BLP. In Sect. 3, we recapitulate the main ideas of adaptive discretization-based algorithms for solving these problem classes. We then introduce in Sect. 4 an open-source C++ library, called libDIPS, which contains implementations of the algorithms reviewed in Sect. 3. In Sect. 5, we compile a library of test problems consisting of unified existing test sets and other test problems found in the literature. Section 6 covers performance tests. We use the library of test problems to evaluate the overall performance of the implemented solvers under 'best-case' parameter settings. In addition, we analyze how sensitive each solver is to changes in its parameters. We summarize our work in Section 7 and propose future work.

2 The problem classes SIP, GSIP, ESIP, MINMAX, and BLP

In the following, we state the problem formulations and the notation used for the problem classes (G)SIP, ESIP, MINMAX, and BLP. Throughout the manuscript, we use bold font for vector-valued symbols and calligraphic font for sets. While we highlight important aspects of each problem class, we discuss only the problem classes and their formulations that are directly tractable with libDIPS. For example, the algorithm for *pessimistic* bilevel problems from [139] can be easily implemented in the discussed software framework; however, we do not discuss this problem class because the algo-

Table 1 Overview of solvers with publicly available implementations for the (deterministic) hierarchical optimization problems supported in libDIPS. We exclude solvers for the purely discrete case and focus on those that allow for nonlinearity or nonconvexity. The solvers by [4, 12] are published but not publicly available. *: Type describes the problem type of the upper-level problem and lower-level problem, respectively. †: Continuous upper-level variables should not appear in the lower level

	SIP	GSIP	BLP	Det. Global	Type (upper-lower)*	Algorithm Type
EAGO [140]	✓			✓	MINLP-MINLP	Adaptive discretization
NSIPS [133, 134]	✓				NLP-unconstrained NLP	Discretization, transcription to integral
SIP with QP-LL [20]	✓			✓	convex NLP-QP	Adaptive discretization and Dualization
IbexSIP [83]	✓			✓	NLP-NLP	Interval Method with Branch & Bound
MibS [126]			✓	✓	MILP-MILP†	Branch & Cut
Fischetti [37]			✓	✓	MILP-MILP†	Branch & Cut
BASBL [71, 96]			✓	✓	MINLP-MINLP	Branch & Sandwich
B-POP [5]			✓	✓	MIQP-MIQP	Multi-parametric
BiOpt [147]			✓		NLP-NLP	Semismooth Newton
Mixed-Integer Bilevel solver [16]			✓		MIQP-QP†	No-good Cuts & Refinement
GAMS EMP [39]			✓	✓	NLP-convex NLP	KKT-based MPEC
libDIPS (this work)	✓	✓	✓	✓	MINLP-MINLP	Adaptive discretization

rithm is currently not implemented. Furthermore, we focus on features related to the algorithmic ideas introduced in Sect. 3. Most problem classes are well-established; we thus provide only a brief description of the algorithms to lay the foundation for the subsequent discussion and refer to surveys for a more thorough discussion [31, 46, 102, 114]. For the problem class of ESIPs, we refer to [30].

SIPs are formulated as

$$\begin{aligned} &\min_{\boldsymbol{x}\in\mathscr{X}} \; f(\boldsymbol{x}) \\ &\quad \text{s.t. } \boldsymbol{g}^{\mathrm{u}}(\boldsymbol{x},\boldsymbol{y}) \leq \boldsymbol{0},\ \forall\, \boldsymbol{y}\in\mathscr{Y}, \\ &\text{with } \mathscr{X} := \left\{\boldsymbol{x}\in\bar{\mathscr{X}} : \boldsymbol{v}^{\mathrm{iu}}(\boldsymbol{x}) \leq \boldsymbol{0},\ \boldsymbol{v}^{\mathrm{eu}}(\boldsymbol{x}) = \boldsymbol{0}\right\} \\ &\qquad \mathscr{Y} := \left\{\boldsymbol{y}\in\bar{\mathscr{Y}} : \boldsymbol{v}^{\mathrm{il}}(\boldsymbol{y}) \leq \boldsymbol{0},\ \boldsymbol{v}^{\mathrm{el}}(\boldsymbol{y}) = \boldsymbol{0}\right\}, \end{aligned} \tag{SIP}$$

the objective function $f : \mathbb{R}^{n_{\mathrm{x}}} \to \mathbb{R}$, the semi-infinite constraint function $\boldsymbol{g}^{\mathrm{u}} : \mathbb{R}^{n_{\mathrm{x}}} \times \mathbb{R}^{n_{\mathrm{y}}} \to \mathbb{R}^{n_{\mathrm{gu}}}$, non-coupling upper-level equality and inequality constraints $\boldsymbol{v}^{\mathrm{iu}} : \mathbb{R}^{n_{\mathrm{x}}} \to \mathbb{R}^{n_{\mathrm{viu}}}$ and $\boldsymbol{v}^{\mathrm{eu}} : \mathbb{R}^{n_{\mathrm{x}}} \to \mathbb{R}^{n_{\mathrm{veu}}}$, non-coupling lower-level equality and inequality constraints $\boldsymbol{v}^{\mathrm{il}} : \mathbb{R}^{n_{\mathrm{y}}} \to \mathbb{R}^{n_{\mathrm{vil}}}$ and $\boldsymbol{v}^{\mathrm{el}} : \mathbb{R}^{n_{\mathrm{y}}} \to \mathbb{R}^{n_{\mathrm{vel}}}$. We denote the n_{x} upper-level variables by $\boldsymbol{x}$, which consist of $n_{\mathrm{x,c}}$ continuous and $n_{\mathrm{x,i}}$ integer variables. Similarly, the n_{y} lower-level variables $\boldsymbol{y}$ consist of $n_{\mathrm{y,c}}$ continuous and $n_{\mathrm{y,i}}$ integer variables. We define the host sets

$$\bar{\mathscr{X}} := \left\{\left[\boldsymbol{x}^{\mathrm{lb}}, \boldsymbol{x}^{\mathrm{ub}}\right] \cap \left(\mathbb{R}^{\mathrm{n_{x,c}}} \times \mathbb{Z}^{\mathrm{n_{x,i}}}\right)\right\}, \quad \bar{\mathscr{Y}} := \left\{\left[\boldsymbol{y}^{\mathrm{lb}}, \boldsymbol{y}^{\mathrm{ub}}\right] \cap \left(\mathbb{R}^{\mathrm{n_{y,c}}} \times \mathbb{Z}^{\mathrm{n_{y,i}}}\right)\right\}, \tag{1}$$

where the superscripts "lb" and "ub" denote finite upper and lower bounds of the upper- and lower-level variables, respectively. These host sets are used in the definitions of the upper- and lower-level feasible sets, $\mathscr{X}$ and $\mathscr{Y}$, in (SIP); as well as in (to follow) (MINMAX), (GSIP), (BLP), and (ESIP). If continuous variables are present, the lower-level feasible set $\mathscr{Y}$ is a set of infinite cardinality.

While non-compact host sets can theoretically be handled with adaptive discretization-based algorithms under certain assumptions [65] — provided they are supported by the subsolvers — we specifically restrict ourselves to host sets defined by finite bounds on both the continuous and integer variables and assume that all functions involved are continuous within these bounds.

We consider MINMAXes of the form

$$\begin{aligned} &\min_{\boldsymbol{x}\in\mathscr{X}} \max_{y\in\mathscr{Y}} f(\boldsymbol{x},\boldsymbol{y}), \\ &\text{with } \mathscr{X} := \left\{\boldsymbol{x}\in\bar{\mathscr{X}} : \boldsymbol{v}^{\mathrm{iu}}(\boldsymbol{x}) \leq \boldsymbol{0},\ \boldsymbol{v}^{\mathrm{eu}}(\boldsymbol{x}) = \boldsymbol{0}\right\} \\ &\qquad \mathscr{Y} := \left\{\boldsymbol{y}\in\bar{\mathscr{Y}} : \boldsymbol{v}^{\mathrm{il}}(\boldsymbol{y}) \leq \boldsymbol{0},\ \boldsymbol{v}^{\mathrm{el}}(\boldsymbol{y}) = \boldsymbol{0}\right\}, \end{aligned} \tag{MINMAX}$$

the objective function $f : \mathbb{R}^{n_{\mathrm{x}}} \times \mathbb{R}^{n_{\mathrm{y}}} \to \mathbb{R}$ and all other functions and sets as defined for (SIP). (MINMAX) is a specialization of (SIP), as can be seen from the reformulation

$$\begin{aligned} \min_{\boldsymbol{x}\in\mathscr{X},t\in\mathbb{R}} \quad & t \\ \text{s.t. } & f(\boldsymbol{x},\boldsymbol{y}) - t \leq 0,\ \forall\, \boldsymbol{y}\in\mathscr{Y}, \\ \text{with } \mathscr{X} &:= \left\{\boldsymbol{x}\in\bar{\mathscr{X}} : \boldsymbol{v}^{\mathrm{iu}}(\boldsymbol{x}) \leq \boldsymbol{0},\ \boldsymbol{v}^{\mathrm{eu}}(\boldsymbol{x}) = \boldsymbol{0}\right\} \\ \mathscr{Y} &:= \left\{\boldsymbol{y}\in\bar{\mathscr{Y}} : \boldsymbol{v}^{\mathrm{il}}(\boldsymbol{y}) \leq \boldsymbol{0},\ \boldsymbol{v}^{\mathrm{el}}(\boldsymbol{y}) = \boldsymbol{0}\right\}. \end{aligned} \tag{MINMAX-REF}$$

GSIPs are an extension to SIPs that allow dependency of the lower-level feasible set on the upper-level variables. We consider GSIPs of the form

$$\begin{aligned} \min_{\boldsymbol{x}\in\mathscr{X}} \quad & f(\boldsymbol{x}) \\ \text{s.t. } & \boldsymbol{g}^{\mathrm{u}}(\boldsymbol{x},\boldsymbol{y}) \leq \boldsymbol{0},\ \forall\, \boldsymbol{y}\in\mathscr{Y}(\boldsymbol{x}), \\ \text{with } \mathscr{X} &:= \left\{\boldsymbol{x}\in\bar{\mathscr{X}} : \boldsymbol{v}^{\mathrm{iu}}(\boldsymbol{x}) \leq \boldsymbol{0},\ \boldsymbol{v}^{\mathrm{eu}}(\boldsymbol{x}) = \boldsymbol{0}\right\} \\ \mathscr{Y}(\boldsymbol{x}) &:= \left\{\boldsymbol{y}\in\bar{\mathscr{Y}} : \boldsymbol{g}^{\mathrm{l}}(\boldsymbol{x},\boldsymbol{y}) \leq \boldsymbol{0},\ \boldsymbol{v}^{\mathrm{il}}(\boldsymbol{y}) \leq \boldsymbol{0},\ \boldsymbol{v}^{\mathrm{el}}(\boldsymbol{y}) = \boldsymbol{0}\right\}, \end{aligned} \tag{GSIP}$$

the coupling lower-level inequality constraint function $\boldsymbol{g}^{\mathrm{l}} : \mathbb{R}^{n_{\mathrm{x}}} \times \mathbb{R}^{n_{\mathrm{y}}} \to \mathbb{R}^{n_{\mathrm{gl}}}$, and all other functions and sets as defined for (SIP).

ESIPs are another generalization of SIPs. ESIPs can be considered as SIPs with another hierarchical optimization problem embedded within them. In this sense, they can be considered a three-level optimization problem. Similarly to robust optimization problems that can be formulated as SIPs, two-stage-robust optimization problems can be formulated as ESIPs [30, 125]. As the ESIP support in libDIPS is preliminary and we do not present a library of test problems for it, we refer to Sect. A for further details.

The optimistic formulation of a BLP reads

$$\begin{aligned} \min_{\boldsymbol{x}\in\mathscr{X},\boldsymbol{y}\in\bar{\mathscr{Y}}} \quad & f(\boldsymbol{x},\boldsymbol{y}) \\ \text{s.t. } & \boldsymbol{g}^{\mathrm{u}}(\boldsymbol{x},\boldsymbol{y}) \leq \boldsymbol{0} \\ & \boldsymbol{y} \in \arg\min_{z\in\mathscr{Y}(\boldsymbol{x})} h(\boldsymbol{x},\boldsymbol{z}), \\ \text{with } \mathscr{X} &:= \left\{\boldsymbol{x}\in\bar{\mathscr{X}} : \boldsymbol{v}^{\mathrm{iu}}(\boldsymbol{x}) \leq \boldsymbol{0},\ \boldsymbol{v}^{\mathrm{eu}}(\boldsymbol{x}) = \boldsymbol{0}\right\} \\ \mathscr{Y}(\boldsymbol{x}) &:= \left\{\boldsymbol{y}\in\bar{\mathscr{Y}} : \boldsymbol{g}^{\mathrm{l}}(\boldsymbol{x},\boldsymbol{y}) \leq \boldsymbol{0},\ \boldsymbol{v}^{\mathrm{il}}(\boldsymbol{y}) \leq \boldsymbol{0},\ \boldsymbol{v}^{\mathrm{el}}(\boldsymbol{y}) = \boldsymbol{0}\right\}, \end{aligned} \tag{BLP}$$

the upper-level objective function $f : \mathbb{R}^{n_{\mathrm{x}}} \times \mathbb{R}^{n_{\mathrm{y}}} \to \mathbb{R}$, the lower-level objective function $h : \mathbb{R}^{n_{\mathrm{x}}} \times \mathbb{R}^{n_{\mathrm{y}}} \to \mathbb{R}$, and all other functions and sets defined as for (SIP). Note that (BLP) contains variables $\boldsymbol{y}$ which are constrained to be in the set of global optimizers of an embedded optimization problem.

The reviewed problem classes are closely related and, thus, share some common traits. For example, due to their hierarchical structure, establishing the feasibility of a candidate solution point requires solving an embedded optimization problem globally for all classes. Thus, global optimization techniques are (implicitly or explicitly) needed in general for the solution process, and consequently, all the problem classes mentioned are computationally demanding. Since ESIPs and GSIPs are generalizations of SIPs and MINMAXes are a specialization of SIPs, one can, under certain

assumptions, use any algorithm for solving ESIPs or GSIPs to solve the other two problem classes. However, given the expected high computational cost of solving these problems, implementing specialized algorithms is likely beneficial over an implementation that only addresses the most general problem class. BLPs and GSIPs are closely related: if $\nexists \bar{\boldsymbol{x}} \in \mathscr{X}$ such that $\mathscr{Y}(\bar{\boldsymbol{x}}) = \emptyset$, (BLP) and (GSIP) are equivalent [118]. However, there is a distinctive difference if a point $\bar{\boldsymbol{x}}$ exists which leads to an empty lower-level feasible set $\mathscr{Y}(\bar{\boldsymbol{x}}) = \emptyset$; the point $\bar{\boldsymbol{x}}$ is infeasible in (BLP), while it is feasible in (GSIP). The close relation between the problem classes is also reflected in the close connection between the algorithmic approaches discussed in the next section.

3 Adaptive discretization-based algorithms

In this section, we introduce multiple adaptive discretization-based algorithms for the solution of the hierarchical optimization problems covered in Sect. 2, which are implemented in our open-source software libDIPS.

We highlight similarities and connections between algorithms to illustrate the benefits of collecting the implementations of these adaptive discretization-based algorithms in a single software. Note that we only give a basic description of the algorithms, focusing on the essential algorithmic ideas and the algorithm's parameters. We refer to the original publications for a detailed description.

A significant benefit of adaptive discretization-based algorithms is that they can utilize well-established optimization solvers, mainly as a black box. The main challenge in solving (SIP), (GSIP), (ESIP) (ref. Sect. A), (MINMAX), and (BLP) compared to standard nonlinear optimization problems is the infinite cardinality of the set $\mathscr{Y}$. This challenge can be addressed through various generalizations and adaptations of the adaptive discretization-based algorithm for SIPs proposed by Blankenship & Falk [15], which is inspired by [103]. Conceptually, the approach of [15] replaces the infinite index set $\mathscr{Y}$ by a finite discretized set $\mathscr{Y}^{\mathrm{d}} \subsetneq \mathscr{Y}$. This discretized problem gives an approximation of (SIP). Through an adaptive refinement scheme, points are added to $\mathscr{Y}^{\mathrm{d}}$, and the approximation of (SIP) is improved.

The implemented solvers are listed in Table 2. Note that apart from the solver *BLP-Box*, all implemented solvers are capable of handling integer variables. The interested reader may refer to Sect. E, where we motivate that although some of the publications on which we base our implementation do not explicitly cover the integer case, the implemented solvers support integer variables.

All the algorithms implemented as solvers in libDIPS are conceptually closely related to the approach of [15]. Therefore, we will provide a conceptual description of the algorithms within the context of adapting the central algorithmic idea presented in [15]. We refer the interested reader to the original publications listed in Table 2 for a detailed description of the algorithm underlying the respective solver.

Table 2 Overview of the adaptive discretization-based algorithms implemented in libDIPS. While the integer case is not explicitly treated in all original publications, its extension is straightforward in most cases, and our implementation already supports this case, except for *BLP-box*. For an informal argument on the integer capabilities, refer to Sect. E

Problem Class	Solver Name	Original Publication	Comment
SIP	*B&F*	[15]	Finite termination with feasible point not guaranteed
	RRHS	[92]	
	Oracle	[131]	
	Hybrid	[29]	
MINMAX	*Minmax*	[34]	
GSIP	*GSIP-RRHS*	[92]	Finite termination not guaranteed [54]
	*	[28]	Relax GSIP to derive an SIP; solve resulting SIP with any SIP solver *, c.f., Sect. 3.2
ESIP	*B&F*	[30]	Upper-bounding procedure of [30] not implemented
BLP	*BLP-Box*	[90]	Implementation does not support discrete variables; extension presented in [88] supports discrete variables
	BLP-noBox	[28]	Supports discrete variables, but numerical experience is limited

3.1 Approaches for SIPs

Algorithmic approach underlying *B&F*

The SIP algorithm of Blankenship & Falk [15] proposes to relax (SIP) by replacing the infinite index set with a finite one, i.e., $\mathscr{Y}^{\mathrm{LBP}} \subsetneq \mathscr{Y}$. We implement this algorithm in libDIPS as the solver called *B&F*. The resulting upper-level problem

$$\begin{aligned} &\min_{\boldsymbol{x}\in\mathscr{X}} \ f(\boldsymbol{x}) \\ &\quad \text{s.t. } \boldsymbol{g}^{\mathrm{u}}\left(\boldsymbol{x}, \boldsymbol{y}^k\right) \leq \mathbf{0}, \ \forall\, \boldsymbol{y}^k \in \mathscr{Y}^{\mathrm{LBP}}, \\ &\text{with } \mathscr{X} := \left\{\boldsymbol{x} \in \bar{\mathscr{X}} : \boldsymbol{v}^{\mathrm{iu}}(\boldsymbol{x}) \leq \mathbf{0}, \ \boldsymbol{v}^{\mathrm{eu}}(\boldsymbol{x}) = \mathbf{0}\right\}, \end{aligned} \tag{LBP}$$

yields a candidate point $\bar{\boldsymbol{x}}$; if (LBP) is solved globally, a lower bound on (SIP) is obtained. Feasibility of point $\bar{\boldsymbol{x}}$ is checked through solving the lower-level problem

$$\max_{y \in \mathscr{Y}} \max_{j \in \{1 \ldots n_{\mathrm{gu}}\}} g_j^{\mathrm{u}}(\bar{\boldsymbol{x}}, \boldsymbol{y}),$$
$$\text{with } \mathscr{Y} := \left\{ \boldsymbol{y} \in \bar{\mathscr{Y}} : \boldsymbol{v}^{\mathrm{il}}(\boldsymbol{y}) \leq \boldsymbol{0},\ \boldsymbol{v}^{\mathrm{el}}(\boldsymbol{y}) = \boldsymbol{0} \right\}, \tag{LLP}$$

and upper-level variables fixed to the corresponding values.

If the optimal objective value of (LLP) is less than or equal a predefined tolerance $\varepsilon^{\mathrm{a}} \geq 0$, $\bar{\boldsymbol{x}}$ is ε^{a}-SIP-feasible (c.f., Definition 1 in Sect. D.1) and the algorithm terminates. If the optimal objective value is strictly greater than ε^{a}, the solution point $\boldsymbol{y}^*$ of (LLP) is added to the discretization, i.e., $\mathscr{Y}^{\mathrm{LBP}} \leftarrow \mathscr{Y}^{\mathrm{LBP}} \cup \boldsymbol{y}^*$; then, (LBP) is solved again.

Remark 1 Instead of solving (LLP), where we maximize over all entries of $\boldsymbol{g}^{\mathrm{u}}$, it is also possible to solve a separate lower-level problem for each entry. Note that if we solve a separate lower-level problem for each entry of $\boldsymbol{g}^{\mathrm{u}}$, we must solve n_{gu} optimization problems, and up to n_{gu} points are added to the discretization in each iteration.

Another possibility is to introduce for each entry of $\boldsymbol{g}^{\mathrm{u}}$ a separate discretization $\mathscr{Y}^{\mathrm{d},i}$ with $i = 1, \ldots, n_{\mathrm{gu}}$.

libDIPS supports both alternatives, but neither is used in the numerical experiments in Sect. 6.

It has been proven numerous times in literature [15, 29, 65, 89] that if (SIP) is feasible and $\varepsilon^{\mathrm{a}} = 0$, the accumulation points of this algorithm are optimal SIP-feasible points (c.f., Definition 3 in Sect. D.1). Additionally, if (SIP) is infeasible, (LBP) will be infeasible after finitely many iterations. However, there can generally be no finite termination guarantee for $\varepsilon^{\mathrm{a}} = 0$. Finite termination can only be guaranteed for $\varepsilon^{\mathrm{a}} > 0$. However, in this case, an SIP-feasible point is usually not generated. The proof of finite convergence usually relies on relatively mild assumptions, i.e., compactness of host sets, continuity of f and $\boldsymbol{g}^{\mathrm{u}}$ on their respective host sets, and approximate global solution of subproblems [15, 29, 56, 65, 89].

Algorithmic approaches underlying *RRHS*, *Oracle*, and *Hybrid*

Since the approach of *B&F* only generates lower bounds and generally does not terminate finitely with an SIP-feasible point, several adaptations of the underlying algorithm have been proposed to generate improving sequences of upper bounds (UBD). The following algorithms share the commonality that, akin to [15], they iteratively solve a sequence of subproblems: a discretized upper-level problem and, subsequently, a lower-level problem for fixed upper-level variables. The solution points of the lower-level problem are then used to discretize the upper-level problem.

The upper-bounding approaches of the implemented solvers can be summarized as follows: The algorithm proposed by [89], implemented as *RRHS*, achieves finite termination through an upper-bounding scheme in combination with the lower-bounding scheme from *B&F*. The algorithm proposed by [131], implemented as *Oracle*, utilizes an oracle problem to conduct a bisection search in the objective space. The algorithm proposed by [29], implemented as *Hybrid*, attempts to combine the best of both, *RRHS* and *Oracle*.

As mentioned in the introduction, other upper-bounding approaches exist; however, since they are not currently implemented in libDIPS, we will not review them here and instead focus on the mentioned algorithms in the following:

RRHS generates upper bounds through an upper-bounding procedure using a so-called upper-bounding problem. This upper-bounding problem is constructed by relaxing (SIP) by replacing the set $\mathscr{Y}$ with a set of finite cardinality, i.e., $\mathscr{Y}^{\mathrm{UBP}} \subsetneq \mathscr{Y}$ (equivalent to *B&F*). At the same time, the discretized (semi-infinite) constraint is tightened by introducing a restriction of the right-hand side $\varepsilon^{\mathrm{r}} > 0$, which is initialized with the parameter $\varepsilon^{\mathrm{r},0} > 0$, resulting in

$$\begin{aligned} &\min_{\boldsymbol{x} \in \mathscr{X}} \ f(\boldsymbol{x}) \\ &\quad \text{s.t. } \boldsymbol{g}^{\mathrm{u}}\left(\boldsymbol{x}, \boldsymbol{y}^k\right) \leq -\varepsilon^{\mathrm{r}} \cdot \mathbf{1}, \ \forall\, \boldsymbol{y}^k \in \mathscr{Y}^{\mathrm{UBP}}, \\ &\text{with } \mathscr{X} := \left\{\boldsymbol{x} \in \bar{\mathscr{X}} : \boldsymbol{v}^{\mathrm{iu}}(\boldsymbol{x}) \leq \mathbf{0}, \ \boldsymbol{v}^{\mathrm{eu}}(\boldsymbol{x}) = \mathbf{0}\right\}. \end{aligned} \tag{UBP}$$

Note that neither a relaxation nor a restriction of (SIP) is generally attained by (UBP) because of the discretization (relaxation) combined with the restriction of the right-hand side. Therefore, it is not guaranteed that the solution of (UBP) immediately generates an upper-bound. However, the generation of (improving) upper-bounds is guaranteed through a successive finer discretization of $\mathscr{Y}^{\mathrm{UBP}}$ and a decreasing rule for ε^{r}, which is applied under certain conditions. The employed decreasing rule is $\varepsilon^{\mathrm{r}} \leftarrow \frac{\varepsilon^{\mathrm{r}}}{\varepsilon^{\mathrm{red}}}$, with a fixed reduction rate $\varepsilon^{\mathrm{red}}$.

The algorithm alternates between iterations that solve (LBP), as in *B&F*, and iterations that solve (UBP). In both cases, (LLP) is subsequently solved to determine SIP-feasibility of the candidate points generated by either (UBP) or (LBP). The solution points of (LLP) are used to populate $\mathscr{Y}^{\mathrm{LBP}}$ or $\mathscr{Y}^{\mathrm{UBP}}$, respectively.

Finite termination to an ε^{f}-optimal SIP-Slater point (see Definition 7 in Sect. D.3) is guaranteed through the converging lower-bounding procedure, which is equivalent to *B&F*, and the converging upper-bounding procedure using (UBP) with the decreasing rule for ε^{r}. As *RRHS* uses the idea of [15] as the lower bounding procedure, the same assumptions are necessary. The employed upper-bounding procedure additionally relies on the existence of an ε^{f}-optimal SIP-Slater point and a decreasing rule of ε^{r}.

Informally, the primary objective behind the upper-bounding procedure using (UBP) is to identify an appropriate combination of two key factors: a discretization $\mathscr{Y}^{\mathrm{UBP}}$ (which corresponds to a relaxation of the set $\mathscr{Y}$) and a restriction ε^{r} applied to the right-hand side of the semi-infinite constraint. With a suitable combination, SIP-feasible points are generated that also provide an upper bound to (SIP). In short, if (UBP) is feasible, but the corresponding solution is SIP-infeasible, we need to reduce the relaxation by enhancing the discretization. Otherwise, we reduce the restriction ε^{r} since it was either too restrictive and thus led to the infeasibility of (UBP), even with the relaxation inherent in the discretization, or a SIP-feasible point was found, signaling the need to reduce the restriction.

Note that the performance of the upper-bounding procedure is strongly influenced by the initial restriction parameter $\varepsilon^{\mathrm{r},0}$, the rate of reduction $\varepsilon^{\mathrm{red}}$, and the initial dis-

cretization $\mathscr{Y}^{\mathrm{UBP}}$. For example, reducing the restriction too quickly, for instance, by using a large reduction rate $\varepsilon^{\mathrm{red}}$, can lead to many iterations [29].

Oracle uses an oracle adaption of [15] to determine if a target objective value f^{t} is attainable. Given preexisting upper and lower bounds on the optimal objective value of (SIP), the target objective value f^{t} is updated through a bisection search in the objective space. The adapted, discretized upper-level problem is given as

$$\min_{\boldsymbol{x}\in\mathscr{X}} \max\left\{ f(\boldsymbol{x}) - f^{\mathrm{t}}, \max_{\substack{\boldsymbol{y}^k\in\mathscr{Y}^{\mathrm{ORA}},\\ j\in\{1\ldots n_{\mathrm{gu}}\}}} g_j^{\mathrm{u}}\left(\boldsymbol{x}, \boldsymbol{y}^k\right) \right\}, \tag{ORA}$$
$$\text{with } \mathscr{X} \coloneqq \left\{\boldsymbol{x}\in\bar{\mathscr{X}} : \boldsymbol{v}^{\mathrm{iu}}(\boldsymbol{x}) \leq \boldsymbol{0},\ \boldsymbol{v}^{\mathrm{eu}}(\boldsymbol{x}) = \boldsymbol{0}\right\},$$

and $\mathscr{Y}^{\mathrm{ORA}} \subsetneq \mathscr{Y}$. This problem can be reformulated to avoid the max-operator, c.f., (ORA-REF) in Sect. B.1.

Analogous to *B&F*, (ORA) is solved with a subsequent solution of (LLP) with fixed upper-level variables $\bar{\boldsymbol{x}}$. If the optimal objective value of (ORA) is greater than 0, f^{t} is not attainable. If (LLP) proves the current candidate point $\bar{\boldsymbol{x}}$ to be SIP-infeasible, the optimal solution point of (LLP) is used for the discretization of $\mathscr{Y}^{\mathrm{ORA}}$. Else $\bar{\boldsymbol{x}}$ is SIP-feasible, and f^{t} is attainable.

Although one might expect the method to inherit strong robustness from the bisection procedure, *Oracle* relies on stricter assumptions for finite convergence compared to *RRHS*. These stricter assumptions pertain to the local minima of $\boldsymbol{g}^{\mathrm{u}}$ (Lemma 4.2 in [131]).

Hybrid aims to combine the ideas of the two previous approaches. Recall that the performance of *RRHS* depends on the parameters $\varepsilon^{\mathrm{r},0}$ and $\varepsilon^{\mathrm{red}}$. *Hybrid* essentially uses the same iterative approach for the upper- and lower-bounding as *RRHS*. Additionally, the subproblem

$$\begin{aligned} \min_{\boldsymbol{x}\in\mathscr{X},\eta}\ & -\eta \\ \text{s.t. } & f(\boldsymbol{x}) - f^{\mathrm{RES}} \leq 0 \\ & \boldsymbol{g}^{\mathrm{u}}\left(\boldsymbol{x}, \boldsymbol{y}^k\right) \leq -\eta\cdot\boldsymbol{1},\ \forall\, \boldsymbol{y}^k \in \mathscr{Y}^{\mathrm{RES}}, \end{aligned} \tag{RES}$$
$$\text{with } \mathscr{X} \coloneqq \left\{\boldsymbol{x}\in\bar{\mathscr{X}} : \boldsymbol{v}^{\mathrm{iu}}(\boldsymbol{x}) \leq \boldsymbol{0},\ \boldsymbol{v}^{\mathrm{eu}}(\boldsymbol{x}) = \boldsymbol{0}\right\},$$

is solved which employs a seperate discretization $\mathscr{Y}^{\mathrm{RES}}$ and target objective value f^{RES}, similarly to f^{t} in (ORA). In this way, some iterations can generate an adaptive optimal update to the restriction parameter ε^{r} by using the solution of (RES) to update ε^{r}. Incorporating the updated restriction parameter in (UBP), *Hybrid* aims to reduce the performance dependence on the parameters $\varepsilon^{\mathrm{r},0}$ and $\varepsilon^{\mathrm{red}}$. Finite convergence of *Hybrid* is guaranteed under the same assumptions necessary for *RRHS* [29].

Table 3 List of subproblems used in the solvers. Note that some of the solvers automatically generate auxiliary problems. Integer variables in the upper and lower levels are supported by all solvers except for the current version of *BLP-Box*. $^{+}$ Subproblem definition see Sect. B.2. † Subproblem definition see Sect. B.4. ‡ Subproblem definition see Sect. A

Problem Class	Solver Name	Subproblems Provided by the User
SIP	*B&F*	(LBP), (LLP)
	RRHS	(LBP), (UBP), (LLP)
	Oracle	(LBP), (ORA), (LLP)
	Hybrid	(LBP), (UBP), (RES), (LLP)
MINMAX	*Minmax*	(MINMAX-LBP)$^{+}$, (MINMAX-LLP)
GSIP	*GSIP-RRHS*	(LBP), (UBP), (GSIP-LLP), (GSIP-AUX)
	*	see SIP solvers
BLP	*BLP-noBox*	(BLP-LBP), (BLP-UBP), (BLP-LLP)†, (BLP-AUX)†
	BLP-Box	(BLP-LBP), (BLP-UBP), (BLP-LLP)†, (BLP-AUX)†, (BLP-AUX-V)†
ESIP	*B&F*	(ESIP-LBP)‡, (ESIP-MLP)‡, (ESIP-LLP)‡

Table 3 summarizes the subproblems of the SIP algorithms above. The user must provide these subproblems to use the respective solver in libDIPS. The subproblems of the respective solvers of the problem classes GSIPs, ESIPs, MINMAXes, and BLPs, covered in the following sections, are also listed in Table 3.

Algorithmic approach underlying *Minmax*

In [34], a specialization of the approach from [15] is presented for (MINMAX). This specialization is based on the fact that the best upper bound for a given upper-level point $\bar{\boldsymbol{x}}$ is provided by the lower-level problem

$$\max_{\boldsymbol{y}\in\mathscr{Y}} f(\bar{\boldsymbol{x}}, \boldsymbol{y}), \quad \text{with } \mathscr{Y} \coloneqq \left\{\boldsymbol{y} \in \bar{\mathscr{Y}} : \boldsymbol{v}^{\mathrm{il}}(\boldsymbol{y}) \leq \boldsymbol{0},\ \boldsymbol{v}^{\mathrm{el}}(\boldsymbol{y}) = \boldsymbol{0}\right\}. \tag{MINMAX-LLP}$$

As a result, an upper bound is readily obtained without the upper-bounding procedures discussed above. A solver using this unique structure is implemented as ***Minmax*** in libDIPS. As *Minmax* is a specialization of *B&F*, it relies on the same underlying assumptions.

3.2 Approaches for GSIPs

There are two main approaches for the global solution of (GSIP) in the context of adaptive discretization-based methods in the literature: (i) relaxation of (GSIP) to an SIP and its solution with an SIP approach and (ii) adaptation of the underlying approach used in *RRHS* to GSIPs.

GSIP to SIP reformulation and relaxation

Commonly, (GSIP) is reformulated and relaxed to an SIP. First, the semi-infinite constraint in (GSIP) is reformulated to

$$\boldsymbol{g}^{\mathrm{u}}(\boldsymbol{x}, \boldsymbol{y}) \leq \boldsymbol{0},\ \forall\ \boldsymbol{y} \in \mathscr{Y}(\boldsymbol{x}) = \{\boldsymbol{y} \in \hat{\mathscr{Y}} : \boldsymbol{g}^{\mathrm{l}}(\boldsymbol{x}, \boldsymbol{y}) \leq \boldsymbol{0}\}, \tag{2}$$

where

$$\hat{\mathscr{Y}} := \left\{\boldsymbol{y} \in \bar{\mathscr{Y}} : \boldsymbol{v}^{\mathrm{il}}(\boldsymbol{y}) \leq \boldsymbol{0},\ \boldsymbol{v}^{\mathrm{el}}(\boldsymbol{y}) = \boldsymbol{0}\right\}. \tag{3}$$

Then, (2) is further reformulated to

$$\boldsymbol{g}^{\mathrm{u}}(\boldsymbol{x}, \boldsymbol{y}) \leq \boldsymbol{0} \vee \boldsymbol{g}^{\mathrm{l}}(\boldsymbol{x}, \boldsymbol{y}) > \boldsymbol{0},\ \forall\ \boldsymbol{y} \in \hat{\mathscr{Y}}, \tag{4}$$

which is subsequently relaxed to

$$\boldsymbol{g}^{\mathrm{u}}(\boldsymbol{x}, \boldsymbol{y}) \leq \boldsymbol{0} \vee \boldsymbol{g}^{\mathrm{l}}(\boldsymbol{x}, \boldsymbol{y}) \geq \boldsymbol{0},\ \forall\ \boldsymbol{y} \in \hat{\mathscr{Y}}. \tag{5}$$

We can now define

$$g_i^{\mathrm{u\ relax}} := \min\left\{g_i^{\mathrm{u}}(\boldsymbol{x}, \boldsymbol{y}), \min_{j \in \{1 \ldots n_{\mathrm{gl}}\}} -g_j^{\mathrm{l}}(\boldsymbol{x}, \boldsymbol{y})\right\}, \tag{GSIP-REF}$$

with $i \in \{1 \ldots n_{\mathrm{gu}}\}$, which corresponds to a min-reformulation of the conjunction in (5). Finally, an SIP relaxation of (GSIP) is obtained, by replacing the left-hand-side of the semi-infinite constraint in (SIP), i.e., $\boldsymbol{g}^{\mathrm{u}}$, with $\boldsymbol{g}^{\mathrm{u\ relax}}$.

The relaxation stems from relaxing the strict inequality in $\boldsymbol{g}^{\mathrm{l}}(\boldsymbol{x}, \boldsymbol{y}) > 0$ to $\boldsymbol{g}^{\mathrm{l}}(\boldsymbol{x}, \boldsymbol{y}) \geq 0$, c.f., (4) and (5). As shown in [48], this relaxation is, under certain assumptions, equivalent to relaxing the feasible set of (GSIP) to its closure. If this property is not present, the relaxation can be strict. In the following, we assume that the relaxation is not strict, i.e., that the minimum of the relaxed problem is equal to the infimum of the original problem [26, 92]. For further details on the GSIP to SIP reformulation and relaxation, the interested reader may refer to [26, 48].

State-of-the-art SIP solvers can be applied, as described in Sect. 3, for the solution of the derived SIP. As it should be apparent whether an SIP or a derived SIP from a GSIP is considered, we use the same solver names as in Sect. 3 whenever a derived SIP is solved with an SIP solver.

Adaption of *RRHS* for GSIPs

The previous approach, i.e., solving the SIP derived from a GSIP with SIP solvers, has the inherent potential disadvantage of ignoring the original GSIP structure. On the contrary, the algorithm we published in [92], implemented as *GSIP-RRHS*, takes this structure into account. Similar to *RRHS*, we [92] employ a restriction of the right-hand side approach to generate upper bounds and an adaptation of *B&F* to generate lower bounds. We use the aforementioned GSIP to SIP relaxation and reformulation, but then account for the GSIP structure when checking the feasibility of a candidate point and when generating discretization points. The main change is that after the GSIP lower-level problem

$$\begin{aligned} g^{\mathrm{u},*} = \max_{\boldsymbol{y}\in\mathscr{Y}} \max_{j\in\{1\ldots n_{\mathrm{gu}}\}} & g_j^{\mathrm{u}}(\bar{\boldsymbol{x}}, \boldsymbol{y}) \\ \text{s.t. } & \boldsymbol{g}^{\mathrm{l}}(\bar{\boldsymbol{x}}, \boldsymbol{y}) \leq \boldsymbol{0}, \\ \text{with } \mathscr{Y} \coloneqq & \left\{\boldsymbol{y} \in \bar{\mathscr{Y}} : \boldsymbol{v}^{\mathrm{il}}(\boldsymbol{y}) \leq \boldsymbol{0},\ \boldsymbol{v}^{\mathrm{el}}(\boldsymbol{y}) = \boldsymbol{0}\right\} \end{aligned} \tag{GSIP-LLP}$$

is solved to check the feasibility of the candidate point $\bar{\boldsymbol{x}}$, the auxiliary problem

$$\begin{aligned} \min_{\boldsymbol{y}\in\mathscr{Y}} \max_{j\in\{1\ldots n_{\mathrm{gl}}\}} & g^{\mathrm{l}}(\bar{\boldsymbol{x}}, \boldsymbol{y}) \\ \text{s.t. } \max_{j\in\{1\ldots n_{\mathrm{gu}}\}} & g_j^{\mathrm{u}}(\bar{\boldsymbol{x}}, \boldsymbol{y}) \geq \alpha \cdot g^{\mathrm{u},*}, \\ \text{with } \mathscr{Y} \coloneqq & \left\{\boldsymbol{y} \in \bar{\mathscr{Y}} : \boldsymbol{v}^{\mathrm{il}}(\boldsymbol{y}) \leq \boldsymbol{0},\ \boldsymbol{v}^{\mathrm{el}}(\boldsymbol{y}) = \boldsymbol{0}\right\} \end{aligned} \tag{GSIP-AUX}$$

and $\alpha > 0$ is solved to find a GSIP-LLP-Slater point, c.f., Definition 6 in Sect. D.2. This GSIP-LLP-Slater point is needed because only a discretization point that strictly fulfills the coupling inequality constraints $\boldsymbol{g}^{\mathrm{l}}(\bar{\boldsymbol{x}}, \boldsymbol{y}) \leq \boldsymbol{0}$ will, if added to the discretization, provide a restriction in the derived subproblem (LBP). The solution point of (GSIP-LLP) will not necessarily provide a restriction if used as a discretization point.

The assumptions used by [92] are similar to those of *RRHS*, but have been adapted for the GSIP case (compactness of host sets, continuity of all functions on their respective host sets, global solution of subproblems, infimum of the original (GSIP) is equal to the minimum of used SIP relaxation, and existence of an ε^{f}-optimal GSIP-Slater point, c.f, Definition 8 in Sect. D.3).

The original publication searched for a GSIP-LLP-Slater point whose relative suboptimality in the lower level was bounded over all iterations by a fixed factor $\alpha > 0$. However, depending on the a priori chosen α, generating a GSIP-LLP-Slater point might fail [54]. Our implementation of this approach, i.e., *GSIP-RRHS*, tries to address the issues raised by [54] by adaptively choosing α whenever a GSIP-LLP-Slater point is not attained by (GSIP-AUX), c.f., Sect. 4.2.1. However, in some instances, numerical issues related to identifying GSIP-LLP Slater points persist for *GSIP-RRHS*, as discussed in Sect. 6.3.

3.3 Approaches for BLPs

For BLPs we implement two solvers, namely ***BLP-Box*** based on [90] and ***BLP-noBox*** based on [28]. The current implementation of *BLP-Box*, based on [90], does not support discrete variables. However, the extension thereof, presented in [88], supports discrete variables. *BLP-Box* and *BLP-noBox* are closely connected to the approach of [15]. This is due to the reformulation of the BLP to a GSIP using the value function reformulation

$$\boldsymbol{y} \in \arg\min_{z \in \mathscr{Y}(\boldsymbol{x})} h(\boldsymbol{x}, z) \iff \boldsymbol{y} \in \mathscr{Y}(\boldsymbol{x}) \wedge \left[h(\boldsymbol{x}, \boldsymbol{y}) - h(\boldsymbol{x}, z) \leq 0,\ \forall\, z \in \mathscr{Y}(\boldsymbol{x})\right].$$

Accordingly, both approaches utilize

$$\begin{aligned} \min_{\boldsymbol{x} \in \mathscr{X},\, \boldsymbol{y} \in \mathscr{Y}(\boldsymbol{x})} \ & f(\boldsymbol{x}, \boldsymbol{y}) \\ \text{s.t. } & \boldsymbol{g}^{\mathrm{u}}(\boldsymbol{x}, \boldsymbol{y}) \leq \boldsymbol{0} \\ & \boldsymbol{x} \in \tilde{\mathscr{X}}\left(\boldsymbol{z}^k\right) \implies h(\boldsymbol{x}, \boldsymbol{y}) - h\left(\boldsymbol{x}, \boldsymbol{z}^k\right) \leq 0,\ \forall\, \boldsymbol{z}^k \in \mathscr{Y}^{\mathrm{LBP}}, \end{aligned}$$

$$\begin{aligned} \text{with } \mathscr{X} &:= \left\{\boldsymbol{x} \in \bar{\mathscr{X}} : \boldsymbol{v}^{\mathrm{iu}}(\boldsymbol{x}) \leq \boldsymbol{0},\ \boldsymbol{v}^{\mathrm{eu}}(\boldsymbol{x}) = \boldsymbol{0}\right\} \\ \mathscr{Y}(\boldsymbol{x}) &:= \left\{\boldsymbol{y} \in \bar{\mathscr{Y}} : \boldsymbol{g}^{\mathrm{l}}(\boldsymbol{x}, \boldsymbol{y}) \leq \boldsymbol{0},\ \boldsymbol{v}^{\mathrm{il}}(\boldsymbol{y}) \leq \boldsymbol{0},\ \boldsymbol{v}^{\mathrm{el}}(\boldsymbol{y}) = \boldsymbol{0}\right\}, \end{aligned} \tag{BLP-LBP}$$

to generate a lower bound based on the discretization $\mathscr{Y}^{\mathrm{LBP}}$, and a parametric set $\tilde{\mathscr{X}}$. The primary difference between the two approaches lies in how the set $\tilde{\mathscr{X}}$ is selected. In [90], the authors compute boxes in each iteration k around the current iterate for the upper-level variables $\bar{\boldsymbol{x}}$. The initial size of the box is proportional to the difference between the upper and lower bounds of the lower-level variable, scaled by a scalar d. This scalar is initialized to d^0 and iteratively reduced by the factor d^{red} until it can be assured that the discretization point $\boldsymbol{z}^k$ stays feasible concerning the lower-level constraints when the upper-level variable $\boldsymbol{x}$ is inside the box. This approach to compute the parametric set $\tilde{\mathscr{X}}$ is implemented in our solver *BLP-Box*. The approach in [28] generates the set $\tilde{\mathscr{X}}$ in a parametric way by setting $\tilde{\mathscr{X}}(z) := \{\boldsymbol{x} \in \mathscr{X} : z \in \mathscr{Y}(\boldsymbol{x})\}$. To generate discretization points, a Slater point of the lower level with respect to the coupling constraint $\boldsymbol{g}^{\mathrm{l}}$ is searched, similarly to the procedure in *GSIP-RRHS* but with an absolute tolerance instead of the relative tolerance α in (GSIP-AUX).

Both approaches utilize a probing problem specifically catered toward BLPs to generate upper bounds. The probing problem is given by

$$\begin{aligned} \min_{\boldsymbol{y} \in \mathscr{Y}(\bar{\boldsymbol{x}})} \ & f(\bar{\boldsymbol{x}}, \boldsymbol{y}) \\ \text{s.t. } & \boldsymbol{g}^{\mathrm{u}}(\bar{\boldsymbol{x}}, \boldsymbol{y}) \leq \boldsymbol{0} \\ & h(\bar{\boldsymbol{x}}, \boldsymbol{y}) \leq h^*(\bar{\boldsymbol{x}}) + \varepsilon^{\mathrm{l,UBP}} \\ & LBD \leq f(\bar{\boldsymbol{x}}, \boldsymbol{y}), \end{aligned} \tag{BLP-UBP}$$

$$\text{with } \mathscr{Y}(\bar{\boldsymbol{x}}) := \left\{\boldsymbol{y} \in \bar{\mathscr{Y}} : \boldsymbol{g}^{\mathrm{l}}(\bar{\boldsymbol{x}}, \boldsymbol{y}) \leq \boldsymbol{0},\ \boldsymbol{v}^{\mathrm{il}}(\boldsymbol{y}) \leq \boldsymbol{0},\ \boldsymbol{v}^{\mathrm{el}}(\boldsymbol{y}) = \boldsymbol{0}\right\},$$

the estimate $h^*(\bar{\boldsymbol{x}})$ for the optimal value of the lower level at the current value for the upper-level variables $\bar{\boldsymbol{x}}$, a small positive tolerance $\varepsilon^{\mathrm{l,UBP}}$, and a known lower bound

on the upper-level objective LBD. The lower-level problem of *BLP-noBox*, *BLP-Box* to compute $h^*(\bar{\boldsymbol{x}})$ is formulated as

$$h^*(\bar{\boldsymbol{x}}) = \min_{\boldsymbol{y} \in \mathscr{Y}(\bar{\boldsymbol{x}})} h(\bar{\boldsymbol{x}}, \boldsymbol{y}),$$

$$\text{with } \mathscr{Y}(\bar{\boldsymbol{x}}) := \left\{ \boldsymbol{y} \in \bar{\mathscr{Y}} : \boldsymbol{g}^{\mathrm{l}}(\bar{\boldsymbol{x}}, \boldsymbol{y}) \leq \boldsymbol{0},\ \boldsymbol{v}^{\mathrm{il}}(\boldsymbol{y}) \leq \boldsymbol{0},\ \boldsymbol{v}^{\mathrm{el}}(\boldsymbol{y}) = \boldsymbol{0} \right\}. \tag{BLP-LLP}$$

Both approaches rely on similar assumptions: compactness of host sets, continuity of all functions on their respective host sets, (approximate) global solution of subproblems, and a Slater-point assumption on the lower-level to be able to construct the discretization, c.f., Assumption 3 in [90] and Assumption 7 in [28]. The assumptions in [28] differ slightly from those in [90], allowing for an approximate solution of the subproblems, discrete variables, and coupling equality constraints that couple the upper and lower level.

4 libDIPS – discretization-based semi-infinite and bilevel programming solvers

As outlined in the introduction and highlighted in Sect. 3, the reviewed algorithms and implemented solvers are closely related and, hence, can benefit from a shared infrastructure. By combining all these solvers into a single software, we can compare them more fairly because they use the same programming language, share a common codebase, and access subsolvers through a unified interface. Therefore, we implemented all solvers in the open-source C++ software libDIPS – Discretization-based semi-Infinite and bilevel Programming Solvers, which was already used in a preliminary form in [26].

The latest version of libDIPS, including the text-based parser, support for the MPI parallelized version of MAiNGO [17], and interfaces to the supported subsolvers, is open-source under the Eclipse Public License v2.0, accessible under https://git.rwth-aachen.de/avt-svt/public/libdips. Additional information on downloading and compiling libDIPS and its dependencies, as well as a list of tested platforms for Microsoft Windows and Linux, is available on the documentation page of the https://git.rwth-aachen.de/avt-svt/public/libdips.

4.1 User-oriented introduction and overview

We present the software structure at a high level in Figure 1, focusing on the user's perspective. Based on this structure, we summarize the high-level workflow of the software:

– The respective subproblems are provided by the user in an easily editable and human-readable form using the domain-specific language provided by libALE [151].

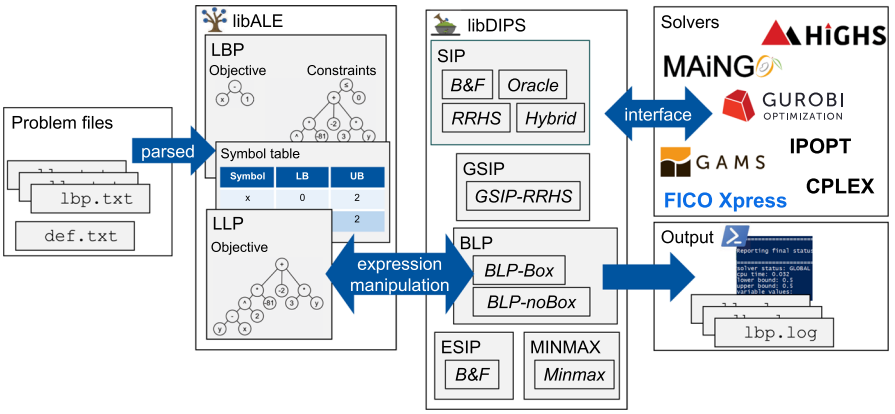


Fig. 1 High-level user interface-oriented software structure overview

- The user input is parsed using libALE, which stores the objective and constraints of each subproblem as logical expression trees and the parameters and variables in a symbol table.
- One of the solvers implemented in libDIPS, c.f., Table 2, which the user has selected, is used to solve the problem.
- One of the several interfaced state-of-the-art optimization solvers is used to solve the subproblem. Currently, CPLEX [62], GUROBI [51], IPOPT [136], FICO Xpress [33], and MAiNGO [17] as well as the GAMS C++ API [39] are available. The user can switch between them easily. Note that, with a suitable license, GAMS provides access to over 30 solvers, including, e.g., BARON [109] and ANTIGONE [85].
- Output is printed onto the command line, and additional log files are written.

There are two options for formulating a problem for use in libDIPS. A significant design decision was to allow the user to manually and individually formulate the subproblems (in an opinionated way). Alternatively, we also provide templated input files that allow for automatically deriving the necessary subproblems from a high-level problem description. In the following Sect. 4.1.1, we elaborate on the benefits of individually formulating the subproblems with a small example.

4.1.1 Example solution of an SIP via *B&F*

Suppose we want to solve the following SIP using *B&F*.

$$\begin{aligned} &\min_{\boldsymbol{x}\in\mathscr{X}} \; f(\boldsymbol{x}) := 2x_1^2 + x_2^2 \\ &\text{s.t. } g^{\mathrm{u}}(\boldsymbol{x}, y) := \min\{x_1 + y_1, x_2 + y_2\} \leq 0, \; \forall\, y \in \mathscr{Y}, \\ &\text{with } \mathscr{X} := \left\{\boldsymbol{x} \in \bar{\mathscr{X}} := [-3, 3]^2 : v^{\mathrm{eu}}(\boldsymbol{x}) := x_1 + x_2 - 1 = 0\right\} \\ &\qquad \mathscr{Y} := [-1, 1]^2 \end{aligned} \tag{Ex}$$

```
definitions:
real[2] x in [−3, 3];

set{index} lbp_k_disc := {};
real[0,2] lbp_y_disc := 0;

objective:
2∗x[1]^2+x[2]^2;

constraints:
x[1]+x[2] = 1.0;
forall k in lbp_k_disc :
    min(x[1]+lbp_y_disc[k,1],x[2]+lbp_y_disc[k,2]) <= 0;
```

Listing 1 Listing of the file `lbp.txt` for the solution of the example problem (Ex). $\mathscr{Y}^{\mathrm{LBP}}$ is represented by `lbp_y_disc` and `lbp_k_disc`, where the former contains the elements of the set and the latter contains the indices. Both are internally updated by the used solver. Note that in this example, $\mathscr{Y}^{\mathrm{LBP}}$ is initialized as an empty set; thus, the size of the first dimension of `lbp_y_disc` is zero.

Using the domain-specific language provided by libALE, we can write the lower-bounding subproblem as a direct transcription of (LBP), as shown in Listing 1 for the `lbp.txt` file. The syntax for the subproblems should be intuitive. For a detailed introduction to the domain-specific language provided by libALE and the syntax for defining an optimization problem, the reader may refer to the libALE documentation [151] and MAiNGO documentation [17].

If we were to naively or automatically generate the subproblem (LLP) from the problem description (i.e., from f, g^{u}, v^{eu} and $\mathscr{Y}$), the resulting objective would read $\max_{\boldsymbol{y}\in\mathscr{Y}} \min\{x_1 + y_1,\ \ x_2 + y_2\}$. Because the user can define the subproblems independently from each other in libDIPS, we can use a different formulation in (LLP). Indeed we should use the epigraph formulation $\max_{\boldsymbol{y}\in\mathscr{Y}} t$ s.t. $t \leq x_1 + y_1,\ t \leq x_2 + y_2$. This formulation enables us to solve a linear optimization problem rather than a nonlinear one. From our experience, such an opinionated reformulation is paramount for an efficient solution in bigger optimization problems, especially when binary variables are involved. The opinionated subproblem formulation of (LLP), transcribed in the `llp.txt` file, is shown in Listing 2.

In addition to the subproblem files, the user has to provide a definitions file, called `def.txt`, which documents the meanings of some of the symbols used in the subproblems. For *B&F*, the user must provide the algorithm with the lower-level variable name, in this case, `y`, and the discretized semi-infinite set of the lower-bounding problem, `lbp_y_disc`, and the index set `lbp_k_disc` of the lower-bounding problem.

The interested reader may refer to Section S1 of the Supporting Information for the necessary subproblem files and extended `def.txt` file for the solution of (Ex) with *RRHS*. Further example input files for the other solvers are provided in the https://git.rwth-aachen.de/avt-svt/public/libdips.

```
definitions:
real t in [−4,4];
real[2] y in [−1, 1];

objective:
−t;

constraints:
x[1]+y[1] >= t;
x[2]+y[2] >= t;
```

Listing 2 Listing of the file `llp.txt` for the solution of (Ex). Note that all subproblems are written as minimization problems.

```
programdefinitions("lbp"):
set_disc(1) = lbp_k_disc;
set_disc_parameter_src(1, lbp_y_disc) = y;
```

Listing 3 Listing of the file `def.txt` for the solution of (Ex) using *B&F*.

4.1.2 Supplied solver instantiations

As the name suggests, libDIPS constitutes a *library* of adaptive discretization-based solvers that can be used to solve hierarchical optimization problems in a C++ application. For convenience, we include a demo executable for each implemented solver listed in Table 2. The inputs to these demo executables are paths to the subproblem files discussed in Sect. 4.1.1 and a dedicated output folder. Other parameters, such as the naming scheme of the input files, are set to default values but can be easily changed in the demo source files. By default, MAiNGO is used as the subsolver. This behavior can be changed by adjusting the corresponding options in the build configuration file.

We also provide the necessary executable to reproduce the computational experiments using the library of test problems, to be introduced in Sect. 5. Additionally, we provide a command-line tool that facilitates similar computational experiments by running the appropriate implemented algorithms for a user-defined list of problem instances.

Please refer to the continuously updated libDIPS documentation for more details on how to set up libDIPS, change the default settings or subsolver, and solve a problem or (a subset of) all test problems using the demo executables of the respective solvers.

4.2 Implementation of the framework

Following the workflow described in Sect. 4.1, the requirements for the framework and structure of libDIPS are

(i) Provision of an intuitive and expressive way for users to input the problem.

(ii) Representation of the problems in a way that facilitates the efficient implementation of the specialized operations needed by the adaptive discretization-based algorithms.
(iii) Interfacing the representation of the subproblems with several subsolvers.
(iv) Abstraction of frequent operations and further auxiliary functionalities, such as logging.

Concerning (i), we provide a simple parser for the metadata written in the `def.txt` file, which *connects* the different subproblems. For the definitions of the subproblems themselves, we decided to use the human-readable domain-specific language provided and parsed by libALE. libALE is also being developed at our chair and was created to be used as a dependency in libDIPS and the deterministic global optimizer MAiNGO. Defining our own domain-specific language enables us to introduce specialized language constructs that aid in formulating hierarchical optimization problems, leading to a more intuitive user experience and supporting features of subsolvers. In terms of intuitive user experience, libALE directly supports universal quantifications over finite sets such as $\forall \boldsymbol{y} \in \mathscr{Y}^{\mathrm{LBP}}$ as a first-class concept. These universal quantifications appear frequently in the subproblems. In the future, we plan to support quantifiers over sets with infinite cardinality, allowing for the natural formulation of hierarchical optimization problems without explicitly formulating the subproblems, thereby eliminating the need for the templates we currently provide. Instead, the subproblems would be automatically derived, providing opportunities for automatic reformulations and presolving. This planned feature is conceptually possible, but it has not been implemented in libALE at present. Regarding the support of subsolver features, libALE already supports certain thermodynamic functions as intrinsic functions, enabling specialized convex relaxation in supporting solvers such as MAiNGO, c.f., [93]. Lastly, using our own domain-specific language also has the benefit that it can be directly customized and further developed to meet the demands of libDIPS and MAiNGO.

Requirement Item (ii) holds importance because the algorithms reviewed in Sect. 3 all follow the same principle: First, single-level subproblems derived from the original (multi-level) problem are defined. Then, these single-level subproblems are iteratively and repeatedly solved using existing optimizers employed as subsolvers. The single-level subproblems are manipulated in each iteration, depending on the solution of the previous subproblem(s). Therefore, there is a significant overlap between the algorithms in terms of the specialized need for subproblem manipulations and operations, e.g., fixing variables, extending the discretization sets, and increasing the dimension of variables. This overlap demands a customizable and general data structure. In the case of libDIPS, the representation and manipulation of algebraic and logical conditions are embodied through libALE. libALE can represent real, index, or boolean-valued expressions and supports sets and tensors of these types. All expressions are stored as expression trees, where nodes are either operations or symbols. The latter includes constant values or named symbols. Named symbols are dereferenced through a symbol table, which links them with the correct values, i.e., parameter values or variables. This allows us to dynamically adjust parameter values or add elements to the discretization sets by using a modified symbol table while keeping a constant expression representation of the subproblem itself.

In addition, the separation of the problem structure from the referenced values through a symbol table also allows for many common tasks in discretization algorithms to be implemented only once because the algorithms primarily interact with the subproblems through high-level operations on the symbol table. Using the example in Listing 1 and Listing 2 of Sect. 4.1.1 in the context of *B&F*, the expressions comprising the subproblems are not changed between iterations. Instead, the discretization of the set $\mathscr{Y}^{\mathrm{LBP}}$ in (LBP), which corresponds to the symbol `lbp_y_disc`, is realized through modifying the set-valued entry of the dereferenced symbol `lbp_y_disc` and `lbp_k_disc` in the symbol table. The dereferenced symbol `lbp_y_disc` is populated with the solution of (LLP), and the set `lbp_k_disc` is expanded by one element. Similarly, the upper-level variables in (LLP) can be fixed while using the same variable names as in (LBP). Fixing variables is achieved by a scope-wise definition of the symbol `x` as a variable in the scope of Listing 1 and as a parameter in the scope of Listing 2. Instead of keeping modified copies of the symbol table for each scope, we support scoped changes, which are thus reversible. Additional shared tasks include updating parameters (such as the restriction parameter ε^{r}) or setting initial values for the variables.

For requirement (iii), i.e., providing an abstract interface for external optimization solvers (subsolvers), libDIPS utilizes the (sub)problem description as expression trees provided by libALE. As described in Sect. 4.1, libDIPS interfaces several state-of-the-art optimization solvers with vastly different internal problem representations. The conversion of the subproblems from their internal representation in libALE to the respective solver format is realized through a solver-specific parser that utilizes a visitor pattern.

Concerning (iv), more advanced visitor patterns are also implemented for auxiliary functionality, such as printing expressions or for categorizing the problem types, c.f., Sect. 5.

4.2.1 Implementation changes of the original algorithms in libDIPS

As revisited in Sect. 2, the implemented algorithms are conceptually closely related. Hence, we were able to enhance the implemented solvers by leveraging ideas from the publications of other algorithms. For example, some of the original publications assume that the subproblems are solved exactly, i.e., the original algorithm statements do not account for the fact that nonlinear problems may only be solved with a given tolerance, while others do.

Additionally, we extended the solvers to, e.g., be able to handle multiple semi-infinite constraints. Section S2 of the Supporting Information provides a detailed overview of the improvements of the implemented algorithms in libDIPS.

5 Library of test problems

As mentioned in the introduction, one of the primary objectives of this manuscript is to compare the different algorithms across a more comprehensive range of problem instances. For this task, we collected problem instances from the litera-

ture of the different problem classes. All problem instances are available in the https://git.rwth-aachen.de/avt-svt/public/libdips. The vast majority of the collected problem instances are academic examples. Nonetheless, the collected examples contain numerous challenging problems that the respective authors specifically chose to highlight and pinpoint specific problems in existing/previous algorithms.

With libDIPS, we provide easy-to-use software for solving hierarchical optimization problems, allowing real-life problem instances to be implemented and, in the future, incorporated into the library of test problems. We encourage such submission to the libDIPS repository. Further, users can easily compare new approaches against the algorithms in libDIPS on the same machine for problems they have selected from the library of test problems. We provide a command-line tool that facilitates such a comparison by running the appropriate implemented algorithms for a user-defined list of problem instances, as described in Sect. 4.1.2.

We want to emphasize that, while this library of test problems is the most comprehensive used in this field, particularly for SIPs, our intention is not to establish it as a fixed benchmark for comparing hierarchical optimization solvers that employ different solution approaches. While our collection of problems, which have already appeared as test problems in the literature, can serve as a basis for creating a benchmark set, it should be expanded through community contributions and carefully curated, similar to the work in more established fields, e.g., [41]. More specifically, creating a proper benchmark test set would require both submissions of real-life instances and an understanding of how the problem type, structure, and formulation interact with different solution approaches. Ideally, this would be a collaborative effort of multiple research groups working on different solution approaches, and ideally include a standard problem file format. Unfortunately, this is outside the scope of this work.

In several cases, the problems could be reformulated to be more suitable for solving with our framework. We did not exploit such individual opinionated problem formulations within the library of test problems; instead, we attempted to keep the problem formulations as close as possible to the initially published problem formulation. We believe this will make it easier to reuse the presented problem instances in other/future works that may not allow for opinionated problem formulations.

However, there are several main changes that we made during problem instance implementation, as we

- added appropriate (sufficiently large) variable bounds if none were provided.
- performed trivial transformations to fit the problem instances in our template, e.g., from minimization to maximization problems.
- performed trivial reformulations to avoid division by zero.
- made reasonable assumptions in case of ambiguous notation or typos.
- replaced open or half-open host sets by closed host sets because all implemented solvers assume closed host sets.
- normalized minmax approximation problems in the MINMAX category to use the same squared objective. Specifically, $\min_{\boldsymbol{x}} \max_{\boldsymbol{y}} |e(\boldsymbol{x}, \boldsymbol{y})|$ was changed to the equivalent formulation $\min_{\boldsymbol{x}} \max_{\boldsymbol{y}} (e(\boldsymbol{x}, \boldsymbol{y}))^2$ to be consistent with problems with

ambiguous formulation in the original literature and to avoid creating artificial duplicates. The latter formulation is chosen as it is preferable for the used subsolver.

We noted further non-trivial changes as supplementary information in the `def.txt` file of the corresponding problem instances.

We employed a two-step heuristic procedure to avoid duplicate problem instances in the presented library of test problems. First, we automatically searched for potential duplicates, i.e., problem instances with the same number of variables, and close optimal objective values or solution points. Second, we manually compared the potential duplicates. When we found a duplicate, we named the problem instance according to the older published source and added the newer source as supplementary information to the `def.txt` file. For the library of BLP test problems, we automatically checked for symbolic equivalence of the objective functions to narrow down the number of potential duplicates in the first step. We also marked problems that are nearly duplicates as similar, e.g., when the problem instances are almost identical but differ only by an additional constraint in the upper or lower level, or if the problem instances were created by a parametric formulation, e.g., in the number of variables. Nevertheless, both problem instances were added to the library of problem instances, even if the change did not change the optimal solution, as equivalence is not straightforward to derive from the problem statement, and these changes might influence the performance of the algorithms. We proceeded in the same manner when an original problem instance was cited, but the presented problem instance was different. For similar problem instances, we added a remark in the corresponding `def.txt` files. We ensured that the similarity relationships expressed by these remarks are transitive and symmetric.

For SIPs, we started with the existing test sets from [89, 134, 138]. Additionally, we have included problem instances from various publications. Table 4 shows a complete list of the used publications. Note that in Table 4 and all following listings, only the oldest found publication with the implemented problem instance is listed. After removing duplicates, the library of test problems contains 336 SIPs; for a complete list of implemented problem instances, see Table S2 in Section S5 of the Supporting Information.

For MINMAXes, we collected problem instances from numerous publications. After removing duplicates, the library of test problems contains 83 problem instances of various publications; for a complete list of implemented problem instances and summary of used publications, see Table 4 and Table S3 in Section S5 of the Supporting Information, respectively.

For GSIPs, we started with the existing test set of [92] and added problem instances from GSIPLib&Gen [112]. The problem instances taken from [112] were reformulated to be compatible with our framework. We also added additional problem instances from various other publications. This led to a total of 86 problem instances after removing duplicates; for a complete list of implemented problem instances and summary of used publications, see Table 4 and Table S4 in Section S5 of the Supporting Information, respectively.

For BLPs, we started with the problem instances from [91] and combined them with the existing benchmark test set BASLib [96] and BOLIB v2 [148]. For some of

Table 4 List of publications used in the library of test problems

Problem Class	Used Publications (Oldest Found)
SIP	[1, 6, 9, 11, 13–15, 18–20, 23, 25, 35, 38, 40, 42, 45, 49, 50, 52–54, 60, 61, 72, 74–76, 78–82, 84, 87, 91, 94, 95, 97–101, 107, 108, 110, 111, 115, 117, 120, 123, 124, 127, 128, 130, 131, 134, 137, 138, 142–146]
GSIP	[6, 21, 25, 54, 55, 63, 69, 74, 77, 79, 92, 104–106, 110, 112, 113, 116, 117, 119, 121, 122, 129, 132, 135]
BLP	[90, 96, 148]
MINMAX	[2, 3, 7, 24, 25, 32, 36, 40, 44, 49, 57–59, 61, 84, 94, 144, 146]

the problem instances, we observed that there are constraints implemented as part of the lower-level feasible set $\mathscr{Y}(\boldsymbol{x})$ but only depend on the upper-level variables $\boldsymbol{x}$. We assumed that the original intent was to include these constraints as upper-level ones. In these cases, we moved these constraints to the definition of the upper-level feasible set. Our motivation for this transformation is that these constraints can be trivially detected; without it, the solvers encounter numerical issues or performance losses. Additionally, this detail does not change the feasible set of individual problem instances. After removing duplicates, we obtained 167 problem instances by combining the existing benchmark test sets.

We categorized all problem instances according to the problem class of their respective upper- and lower-level problems using the following categories

- *LP* – linear objective function subject to linear constraints
- *QP* – quadratic objective function subject to linear constraints
- *QCQP* – quadratic objective function subject to quadratic constraints
- *NLP* – all problem instances not fitting the aforementioned problem categories

Note that we do not differentiate in terms of convexity because the convexity of the subproblems can vary from iteration to iteration. For example, consider the expression xy^2 in the context of the semi-infinite constraint in (SIP) with x being an upper-level variable and y being a lower-level variable. The convexity with respect to the lower-level variables y depends on the sign of the fixed upper-level variable, which might change in each iteration. The resulting composition of the library of test problems is shown in Table 5, which also indicates the (small) number of problems that include discrete variables.

6 Numerical experiments

All calculations were conducted on the RWTH High-Performance Computing cluster running Rocky Linux 8. No parallelization was used, and each computation was

Table 5 Composition of the libraries of test problems for the different problem categories. UL = problem class of the upper level, specifically the respective (LBP) with a given (non-empty) discretization, LL = problem class of the lower level with fixed upper-level variables

UL	LL			
	LP	QP	QCQP	NLP
(a) Library of SIP test problems. For a breakdown of the number of SIP test problems with discrete variables, see Table 5b.				
LP	22	22	1	75
QP	1	47	2	21
QCQP	2	12	5	11
NLP	9	3	3	100
(b) Breakdown of the number of SIP test problems included in Table 5a, with discrete variables in the upper/lower level, in that order.				
LP	10|0	3|0	0|0	1|2
QP	0|0	0|0	0|0	0|0
QCQP	0|1	0|0	0|0	0|0
NLP	0|0	0|0	0|0	3|4
(c) Library of MINMAX test problems. The library includes one problem instance with integers in the lower-level (shown in brackets).				
LP	0	1	0	1
QP	0	0	0	0
QCQP	0	13	0	44
NLP	0	3	0	21 (1)
(d) Library of GSIP test problems. The library does not include any integer problem instances.				
LP	11	0	4	4
QP	8	3	6	1
QCQP	6	7	8	8
NLP	1	0	4	15
(e) Library of BLP test problems. The library does not include any integer problem instances.				
LP	31	0	0	0
QP	7	0	0	0
QCQP	18	41	7	0
NLP	2	9	10	42

performed on a single core with an Intel Xeon Platinum 8160 Processor "SkyLake" running at 2.1 GHz with up to 20 Gbit RAM. This precluded the calculation for five large instances from [20], which are not included in the following results. All subproblems are solved using MAiNGO version 0.7.1, configured to utilize CPLEX version 22.1.1 to solve linear or quadratic problems. We use MAiNGO for the numerical experiments for two main reasons: Firstly, as MAiNGO is open-source, we configured the libDIPS compilation file to compile MAiNGO and automatically make it a subsolver. As a result, MAiNGO is our default solver. Secondly, MAiNGO supports trigonometric functions as well as the functions abs, min, and max, which occur in some of the problem instances. For example, BARON does not support trigonometric functions. The maximum CPU time per problem instance was set to 20 min. If a problem instance is not solved within that time, the solution procedure is aborted, and the instance is considered not solved (CPU time is set to infinity). The absolute inequality tolerance `deltaIneq` of MAiNGO was set as small as possible to 1×10^{-9}. In Sect. 6.1, to follow, we shortly discuss the implications of `deltaIneq` being > 0. Note that for some automatically generated subproblems, we use specific fixed settings to speed up the convergence of MAiNGO. For example, to compute an initial upper bound in *Oracle*, the objective function is maximized absent any constraints, and the relative optimality tolerance `rel_tol` of MAiNGO is set to $\max\{\texttt{rel\_tol\_lbp}, 0.5\}$. For other subproblems, users can easily set the optimality tolerances. The specific tolerances used for each solver can be found in Table S1 in Section S3 of the Supporting Information. All other settings of MAiNGO, except for the output settings, have been left at their default values.

Within the manuscript, we primarily focus on time as the relevant quantity, as we investigate global deterministic optimization algorithms. We predominantly show, discuss, and compare the performance of the individual solvers within a problem class utilizing Dolan-Moré profiles; we plot the empirical cumulative distribution functions of the ratio of solve CPU time for a given solver against the virtual best, defined as

$$\textit{CPU time performance ratio} = \frac{\textit{CPU time of problem instance } i}{\textit{CPU time of problem instance } i \textit{ of the fastest solver}}. \tag{PR}$$

Our motivation for using CPU time performance ratio profiles is to minimize the importance of problem instance size.

While CPU time performance ratio profiles are widely used and allow for a comprehensive and visual comparison of different algorithms over larger problem sets, they also have disadvantages. For problems with very small solution times, even minor absolute time differences can result in a significant difference in terms of the CPU time performance ratio. To combat this, all CPU times under 1 second (s) have been rounded up to 1 s as we can not fairly differentiate relative runtime measurements for run times under 1 s. Furthermore, CPU time performance ratio profiles can be sensitive to the addition or removal of considered algorithms. However, since the performance profiles based on absolute run times align with our conclusions, we contend this sensitivity does not affect the validity of our findings. Due to the known disadvantages, performance profiles for other metrics, such as CPU time and the number of subprob-

lems solved, as well as additional plots, are included in Section S4 of the Supporting Information and referenced at the appropriate places in the text.

We briefly investigated the consistency of the measured run times by measuring three repeated runs of *RRHS* on the SIP library of test problems. On average, the standard deviation over the three repeated runs, normalized by the mean, is 0.02 and the maximum of this normalized standard deviation is 0.16. Since we consider the deviations insignificant, we refrain from repeating the computations of the more extensive computational studies, which are discussed later in this section.

We ascertained that there is relatively little overhead outside of the subsolver MAiNGO, i.e., the time spent in our C++ code outside the subsolver is negligible compared to the CPU time spent solving the subproblems by the subsolver. For normal-sized problems, on average 99% of the total computation time (wall clock time) is spent solving the subproblems in MAiNGO.

In the following subsections, we present several numerical experiments examining the performance of the implemented solvers across different problem classes with varying parameters. We use the problem instances from the library of test problems presented in Sect. 5 for the comparison, excluding the large instances from [20] that could not be run due to memory limitations. Since *B&F* terminates with SIP-ε^{a}-feasible points (with $\varepsilon^{\mathrm{a}} > 0$), we exclude *B&F* from the performance tests.

Anticipating the results of the following subsections, in summary, the solvers reliably solve most collected problem instances within their respective problem classes. However, we found that, on the one hand, the library of test problems contained many problem instances that were quickly solvable (global solution found in less than 10 s) for all solvers with the considered parameter values. On the other hand, many problem instances could not be solved by any solver within the time limit.

We briefly investigated whether a reliable prediction regarding problem difficulty, i.e., if a given problem instance is *solved quickly* (solved in < 10 s) by all solvers or remains *unsolved* within the time limit (20 min), can be based on easily observable problem instance characteristics. The considered problem instance characteristics are the subproblem class, the number of variables at each level, and the number of distinct constraint types. We attempted to classify the problem instances based on these characteristics using a decision tree with monotonicity constraints. Unfortunately, this analysis did not produce much insight. This is not surprising, as predicting problem difficulty straightforwardly is already challenging for nonconvex NLPs. It remains to be seen whether more sophisticated analyses of problem properties for nonconvex NLPs (to be developed) will also allow a deeper understanding of the factors related to the complexity of hierarchical optimization problems. The trends that we could derive are

- SIPs with less than four upper-level variables are likely to be solved *quickly* (only 20 % of SIPs with less than four upper-level variables are not solved *quickly* and 16 % of SIPs with more than four upper-level variables are solved *quickly*)
- SIPs where the lower-level is not an NLP are likely to be *solved*; ignoring the 25 unsolved large-scale instances from [20] (4 % of *unsolved* problem instances have a lower-level problem that is not an NLP; less than 1 % of problem instances with a lower-level that is not NLP are *unsolved*)

Table 6 Number of problem instances that are always solved *quickly*, i.e., solved in < 10 s, and problem instances that are *unsolvable* within the time limit across all numerical experiments for the different problem categories, in that order. UL = problem class of the upper level, specifically the respective (LBP) with a given (non-empty) discretization, LL = problem class of the lower level with fixed upper-level variables

UL	LL			
	LP	QP	QCQP	NLP
(a) Library of SIP test problems.				
LP	22|0	18|0	0	45|7
QP	1|0	3|25	1|0	5|2
QCQP	2|0	9|0	4|0	6|2
NLP	7|1	2|0	1|0	33|16
(b) Library of GSIP test problems.				
LP	4|1	0	1|0	3|0
QP	2|1	0|2	2|0	0
QCQP	0|4	1|4	2|1	3|1
NLP	0	0	1|0	3|7
(c) Library of MINMAX test problems.				
LP	0	1|0	0	0
QP	0	0	0	0
QCQP	0	7|0	0	12|7
NLP	0	1|0	0	3|11
(d) Library of BLP test problems.				
LP	5|3	0	0	0
QP	2|1	0	0	0
QCQP	2|5	2|7	2|0	0
NLP	0|1	3|2	2|2	15|4

– GSIPs with less than four upper-level variables are likely to be *solved* (10 % of such GSIP are *unsolved*, 3 % of *unsolved* cases have less than four upper-level variables)

A breakdown by the introduced categorization by subproblem type is shown in Table 6.

We want to emphasize that the summary of our empirical results presented above, as well as the detailed results in the following subsections, depend on both the library of test problems used and the subsolver employed. To the best of our knowledge, the library of test problems used is the largest in the known literature to date. Still, it can not be ruled out that we will obtain different empirical results with another (even more) representative library of test problems. This is especially the case for problems with discrete variables because our library of test problems only includes 24 SIPs and 1 MINMAX problem instances with discrete upper- and/or lower-level variables, c.f., Table 5.

6.1 Definition of feasibility and numerical challenges

For SIPs and GSIPs, we use relatively strict criteria for an upper-level point $\bar{\boldsymbol{x}}$ to be considered feasible. In both problem classes, an upper-level point $\bar{\boldsymbol{x}}$ is only feasible if the objective value at the solution of the lower-level problem is less or equal to zero. In the following experiments, we thus do not consider any explicit tolerance on this constraint. This is in contrast to the case of BLPs, where we allow a corresponding tolerance, c.f., Definition 4 in Sect. D.1, of 1×10^{-4} which is likely unavoidable in the nonconvex case [86].

Consistent with this strict feasibility requirement in the case of SIPs and GSIPs, we set the numerical tolerance for inequality constraints as small as possible in the upper-level problems; recall in MAiNGO this corresponds to `deltaIneq` $\geq 1 \times 10^{-9}$. As discussed in [8] for the case of BLPs, constraint tolerances in the solution of the lower-level problems can significantly impact the solution of the overall problem when nonlinear constraints are considered. We thus also consider this strict constraint tolerance in the lower-level problems.

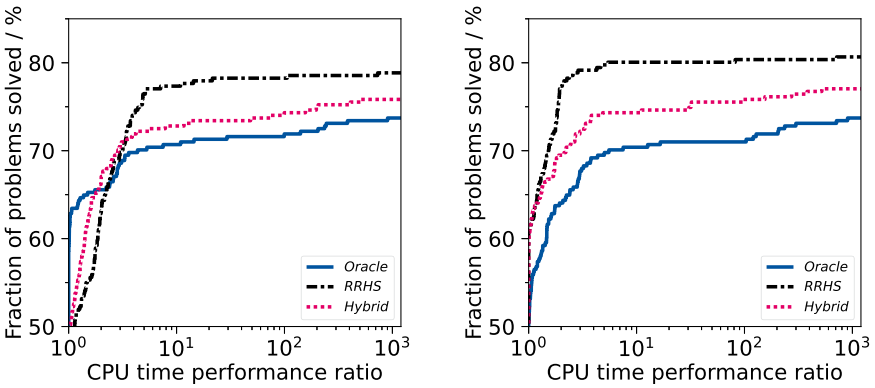
Note that we adaptively refine the optimality tolerances used for the lower level as outlined in [29] for the SIP case, c.f., Section S2 of the Supporting Information. While being necessary, we observed that some numerical issues and long computation times are related to this refinement. For some problems, later iterations require smaller and smaller optimality tolerances in the lower level, as it becomes more likely to be inconclusive, i.e., the computed lower and upper bounds on the optimal objective value of (LLP) bracket zero. In these cases, no statement can be made whether the candidate point $\bar{\boldsymbol{x}}$ of (LBP) is SIP-feasible, c.f., Definition 3 in Sect. D.1.

Introducing looser tolerances bears the possibility of deviating from the rigorous mathematical formulation, which was thus avoided in this work. Still, a user in a specific application should critically evaluate their decision on the chosen tolerances, as they may be overly restrictive compared to the model accuracy and significantly influence the solution time or even solvability.

Another frequent numerical challenge arises due to large coefficients, for example, when working with relatively wide variable bounds in conjunction with large powers. For example, in [142], the following SIP is posed

$$\begin{aligned} \min_{\boldsymbol{x} \in \mathcal{X} = \mathbb{R}^{20}} \ & \boldsymbol{x}^T \boldsymbol{x} \\ \text{s.t. } & g(\boldsymbol{x}, y) = 3 + 4.5 \cos\left(\tfrac{4.7\pi(y-1.23)}{8}\right) - \textstyle\sum_{i=1}^{20} x_i y^{i-1} \leq 0, \quad \forall y \in [0, 200]. \end{aligned}$$

All considered SIP solvers fail for this problem, even after bounding the domain of the upper-level variable to $\mathcal{X} = [-100, 100]^{20}$. The initial upper-level point is $\boldsymbol{x}^0 = \boldsymbol{0}$. If the smallest lower-level optimizer $y^0 \approx 1.23$ is found, some of the approaches converge. However, if another global optimizer of the lower-level problem, e.g., $y^0 \approx 4.63$, is selected, the resulting coefficient of x_{20} is so large that the solution of the next upper-level problem can not be numerically differentiated from $\boldsymbol{0}$, leading to an infinite loop.



(a) "Out of the box performance;" parameters as in the original publications.

(b) Solvers with tuned parameters.

Fig. 2 CPU time performance ratio profiles of SIP solvers: *RRHS*, *Oracle*, and *Hybrid*. Without tuning, a clear trade-off between the fraction of problems solved overall and the fraction of problems solved in the shortest time exists between the solvers. With tuning *RRHS* performs best

6.2 Performance results of SIP solvers

6.2.1 "Out of the box performance"

Figure 2a shows the "out of the box performance" of the SIP solvers with their default parameters, i.e., the parameters of each solver are set to the values of their respective original publication, c.f., Table 7. For completeness, we provide additional performance profiles in the Supporting Information: for CPU time, subproblem number, and subproblem ratio performance performance profiles see Figures S1a, S2a and S2c in Section S4.1, for performance profiles of problem instances with discrete variables in the upper and/or lower level see Figure S7 in Section S4.1.3, for performance profiles of problem instances with purely continuous variables see Figures S5 and S6 in Section S4.1.2. As outlined in Sect. 3, discrete variables do not pose a challenge for the SIP solvers. However, it remains to be seen if this result also holds for a more extensive library of test problems.

Quantitatively, the performance of the solvers on our more extensive library of test problems is similar to the results published in [26]: *Oracle* is the fastest for the largest fractions of problems. Still, when it is not the fastest, it is often significantly slower and is the least robust in the sense that it solves the fewest problems for large CPU time performance ratios within the time limit. In contrast, *RRHS* can solve most problems within the time limit and solves most problems when allowing for CPU time performance ratios exceeding 10. However, in this test, it is the fastest solver for the least number of problems. The solver *Hybrid* can solve significantly more problems than *Oracle* and outperforms *RRHS* for the time-factor range up to factor 3. Overall, the *Hybrid* solver is the most well-rounded with default parameters. However, as shown in Figure 2b, after parameter tuning, *RRHS* is likely the better choice. In the following, we take a closer look at the impact of tuning on two solvers: *RRHS* and *Hybrid*.

Table 7 Parameter values used in the original publications [29, 89, 92, 131] and tuned parameter values. *GSIP-RRHS*: Note that [92] introduces restriction and reduction parameters that may differ between $\boldsymbol{g}^{\mathrm{u}}$ and for $\boldsymbol{g}^{\mathrm{l}}$; however [92] uses the same values for both. $*$: α^{red} is introduced in this work

Problem Class	Solver Name	Paremeters in Original Publication	Tuned Parameters
SIP	*RRHS*	$\varepsilon^{\mathrm{r},0} = 1, \varepsilon^{\mathrm{red}} = 1.5$	$\varepsilon^{\mathrm{r},0} = 0.1, \varepsilon^{\mathrm{red}} = 10$
	Oracle	–	–
	Hybrid	$\varepsilon^{\mathrm{r},0} = 1, \varepsilon^{\mathrm{red}} = 1.2$	$\varepsilon^{\mathrm{r},0} = 0.1, \varepsilon^{\mathrm{red}} = 10$
GSIP	*GSIP-RRHS*	$\varepsilon^{\mathrm{r},0} = 1, \varepsilon^{\mathrm{red}} = 2,$ $\alpha^{0} = 0.5, \alpha^{\mathrm{red}} = *$	$\varepsilon^{\mathrm{r},0} = 1, \varepsilon^{\mathrm{red}} = 2,$ $\alpha^{0} = 0.25\ , \alpha^{\mathrm{red}} = 1.2$
	RRHS	–	$\varepsilon^{\mathrm{r},0} = 5, \varepsilon^{\mathrm{red}} = 5$
	Oracle	–	–
	Hybrid	–	$\varepsilon^{\mathrm{r},0} = 0.1, \varepsilon^{\mathrm{red}} = 5$

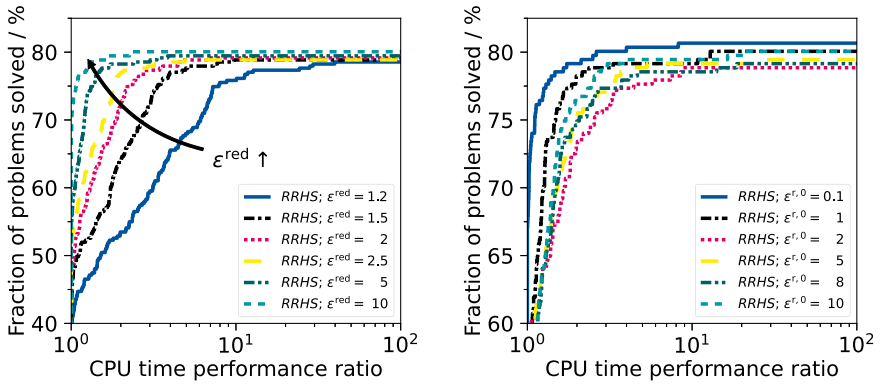
6.2.2 Parameter tuning for *RRHS* and *Hybrid*

Recall that the performance of the upper-bounding procedure, and hence overall performance, of *RRHS* and *Hybrid* is influenced by the initial restriction parameter $\varepsilon^{\mathrm{r},0}$; the rate of reduction, i.e., $\varepsilon^{\mathrm{red}}$; and the initial discretization. Therefore, we performed a parameter study for *RRHS* and *Hybrid* varying the two parameters $\varepsilon^{\mathrm{r},0} = \{0.1, 1, 2, 5, 8, 10\}$ and $\varepsilon^{\mathrm{red}} = \{1.2, 1.5, 2, 2.5, 5, 10\}$. The results of the parameter study are shown in Figures 3 and 4.

RRHS, for a fixed $\varepsilon^{\mathrm{r},0} = 1$ (value in original publication [89]), performs best with a fast reduction of ε^{r}, c.f., Figure 3a. Furthermore, a small initial restriction $\varepsilon^{\mathrm{r},0}$ is beneficial, c.f., Figure 3b. Hence, choosing a big value for $\varepsilon^{\mathrm{red}}$ and a small $\varepsilon^{\mathrm{r},0}$ seems beneficial for the considered library of SIP test problems. With tuned parameters, i.e., $\varepsilon^{\mathrm{red}} = 10$ and $\varepsilon^{\mathrm{r},0} = 0.1$, *RRHS* was able to solve 81% of the problem instances within 20 min. For a detailed listing of the reasons why the remaining problem instances were not solved, refer to Table 8.

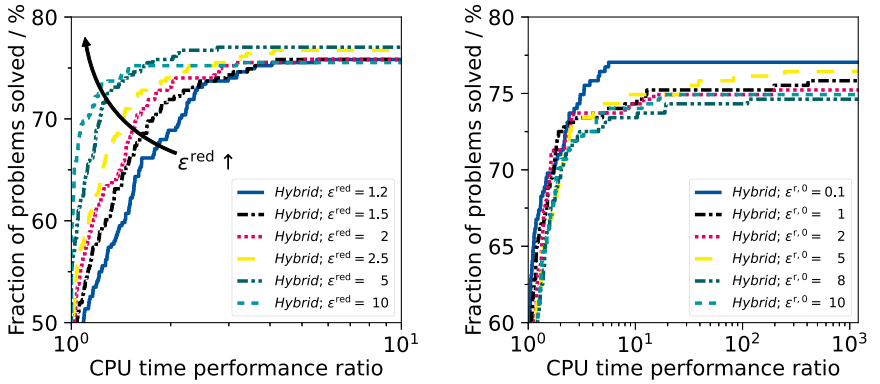
Hybrid, for a fixed $\varepsilon^{\mathrm{r},0} = 1$ (value in original publication [29]), performs best with a fast reduction of ε^{r}, c.f., Figure 4a. Furthermore, a small initial restriction $\varepsilon^{\mathrm{r},0}$ is beneficial, c.f., Figure 4b. With tuned parameters, i.e., $\varepsilon^{\mathrm{red}} = 10$ and $\varepsilon^{\mathrm{r},0} = 0.1$, *Hybrid* was able to solve 77 % of the problem instances within 20 min. For a detailed listing of the reasons why the remaining problem instances were not solved, refer to Table 8.

Figure 2b shows that after parameter tuning *RRHS* outperforms the other solvers (for the CPU time, subproblem number, and subproblem ratio performance profiles see Figures S1b, S2b and S2d in Section S4.1 of the Supporting Information). Furthermore, by comparing the sensitivity of the solvers *RRHS* and *Hybrid* (Figures 3 and 4), we can conclude that *Hybrid* is less sensitive to the choice of the parameters. This is most likely



(a) *RRHS*: $\varepsilon^{\text{r},0} = 1$ and $\varepsilon^{\text{red}} = \{1.2, 1.5, 2, 2.5, 5, 10\}$. (b) *RRHS*: $\varepsilon^{\text{r},0} = \{0.1, 1, 2, 5, 8, 10\}$ and $\varepsilon^{\text{red}} = 10$.

Fig. 3 CPU time performance ratio profiles of the SIP parameter study for *RRHS*. Performance improves with increasing ε^{red}



(a) *Hybrid*: $\varepsilon^{\text{r},0} = 1$ and $\varepsilon^{\text{red}} = \{1.2, 1.5, 2, 2.5, 5, 10\}$. (b) *Hybrid*: $\varepsilon^{\text{r},0} = \{0.1, 1, 2, 5, 8, 10\}$ and $\varepsilon^{\text{red}} = 10$.

Fig. 4 CPU time performance ratio profiles of the SIP parameter study for *Hybrid*. Performance improves with increasing ε^{red}. No systematic behavior is seen for $\varepsilon^{\text{r},0}$

due to the additional subproblem (RES), which is used to generate *optimal* updates for the restriction parameter ε^{r}. However, this update procedure comes with additional computational costs and can not compensate for the reduced computational costs of *RRHS* if the *tuned* parameters are chosen. A reason for the additional computational costs might be that *Hybrid* solves the subproblem (RES), which, beyond being an additional subproblem, might belong to a different problem class than (LBP). For example, if (LBP) is a QP, (RES) becomes a QCQP. However, we also noticed instances where (RES) took much longer than the respective (UBP), even though the upper-level problem was a general nonlinear problem. We were unable to confirm a structural reason for this behavior. In summary, the performance of *RRHS* is superior compared to the other solvers, especially if parameter tuning can be performed.

Table 8 Number of errors encountered in the computational study using tuned parameters, c.f., Table 7. The subsolver errors category encompasses problem instances where MAiNGO failed to solve a subproblem globally or MAiNGO threw an exception. The algorithmic errors category includes problem instances where the maximum number of iterations `max_iter` was exceeded or where algorithmic parameters, e.g., ε^{r}, were reduced below a certain threshold, e.g., `min_eps_res`, c.f., Table S1 in Section S3 of the Supporting Information

Problem Class	Solver Name	Non-Solved / Total	Exceeded max. Time	Subsolver Errors	Algorithmic Errors
SIP	*RRHS*	64/331	33	18	13
	Oracle	87/331	36	41	10
	Hybrid	76/331	35	28	13
GSIP	*GSIP-RRHS*	34/ 86	8	12	15
	RRHS	26/ 86	8	17	1
	Oracle	24/ 86	6	18	–
	Hybrid	26/ 86	6	19	1
BLP	*BLP-noBox*	28/167	6	17	5
	BLP-Box	84/167	31	41	12
MINMAX	*Minmax*	20/ 83	–	20	–

6.2.3 Results on a reduced SIP test set

Our library of SIP test problems comprises numerous problem instances from the literature, and curation is only carried out to a basic level, as described in Sect. 5. In addition to the previously mentioned lack of real-world instances, the results could be skewed by "clusters" of very similar problem instances, which would bias the results in favor of solvers and parameter sets that work exceptionally well for these "clusters."

It is, however, rather difficult to quantify the kind of similarity that would cause such a bias. For example, as previously noted, we observed that adding a single simple constraint to a problem can significantly impact the problem's solution or the solver's behavior. We have already mentioned that the results obtained with standard parameters are qualitatively similar to those obtained on a smaller test set in [26].

Nevertheless, for SIPs, we investigate the stability of the results for solvers with tuned parameters when using a diverse subsample of the problem set, i.e., a reduced test set. For BLPs, we use two established benchmark sets. For GSIPs and MINMAXes, the problem sets are already relatively small, so further reducing the test set would likely impair the validity of the reduced test set.

To create the reduced test set, we first performed a prefiltering step, where we retained only a single representative from a group of problems marked as similar. For problem instances parameterized in the number of variables, only the largest problem instance is considered a candidate in this step. This resulted in 228 problem instances.

Then, we used two approaches to generate a diverse subsample of problem instances. The first is based on k-medoids clustering, utilizing the problem class of upper and lower levels, as well as the number of discrete and continuous variables as features. While the shifted geometric means (with a shift of 10 s) of *RRHS*, *Hybrid*, and *Oracle* over the cluster medoids change when varying the number of clusters, the qualitative results stay consistent after $k = 50$: *RRHS* performs best, *Hybrid* performs slightly worse than *RRHS*, and *Oracle* performs worst. The results for $k = 50$ are shown in Table 9. In the second approach, we used a binning-based subsampling approach. In contrast to the k-medoid approach, which is not suitable for a high number of features, we incorporated additional features: the number of inequality and equality constraints in the respective levels and the number of semi-infinite constraints. We divided the numerical features into five quantiles, assigning each problem to a bin depending on its features. We selected our subsamples by choosing a problem from each bin. For bins containing multiple problems, a single representative was selected randomly. Still, if possible, a problem was chosen that was solved within the time limit by at least one solver in our tests, and for all solvers, more than one second was required to solve it.

From the results displayed in Table 9, we conclude that qualitatively, our ranking of the solvers with tuned parameters is not skewed by clusters of similar problems. We also observe that the inclusion of large-scale instances from [20] significantly alters the geometric means, as most of these problems remain unsolved or require a comparatively long time to solve. Comparing the geometric means over the problems excluding the ones from [20], we see that there is no considerable change with prefiltering, i.e., only considering a single instance from similar problems. One explanation is that, as mentioned, even minor modifications, such as an additional linear constraint in either

Table 9 Shifted geometric means (with a shift of 10 s) of run times over different sets of problems for the different SIP solvers with tuned parameters. Measured times are rounded up to 1 s

	Full Problem Set	Without [20]	Prefiltered Problem Set	50-Medoids	Bin-Based
RRHS	30.4 s	18.2 s	17.8 s	14.3 s	21.3 s
Hybrid	35.4 s	21.6 s	21.2 s	20.2 s	22.7 s
Oracle	44.9 s	28.2 s	27.1 s	31.4 s	38.0 s

level, can change the behavior of the solvers. Additionally, we found that the separate groups of similar problems were diverse, suggesting that there is no overall bias towards a specific structure. *Oracle* seems to perform worse on the reduced problem sets. This appears to support our analysis, indicating that it is especially suitable for specific problems.

6.2.4 Performance on unsolved SIP problem instances

We also investigated the behavior of the SIP solvers with tuned parameters (c.f., Table 7) in problem instances where the solver(s) could not solve the problem instance within the given time limit of 20 min. Suppose a solver is unable to solve a problem instance up to the specified optimality tolerance. In that case, we are interested in the remaining objective value gap. Naturally, bounds on the optimal objective value are preferred over not providing any bounds at all, even if the provided bounds on the optimal objective value are looser than the optimality tolerances used for termination. For problem instances where at least one of the solvers failed to solve the instance within the time limit, we plot in Figure 5 the remaining objective value gap defined as the minimum of the absolute and relative difference between the upper bound (UBD) and lower bound (LBD) of the objective value, i.e.,

$$\text{Gap} = \min\left\{UBD - LBD,\ \frac{UBD - LBD}{UBD}\right\}. \tag{6}$$

Note that no UBD, i.e., feasible solution, is provided by any solver for most problem instances that are unsolved by all considered solvers (with tuned parameters) within the time limit (see Figure S4 in Section S4.1.1 of the Supporting Information).

We can interpret the depicted Gap in Figure 5 in two ways: (i) in terms of reported solution quality, i.e., the gap associated with the solutions produced, and (ii) in terms of sensitivity to the used optimality tolerances, i.e, if the problem instance would have been solved with a higher absolute and relative objective tolerance.

Concerning (i) the reported solution quality, similar to the results over all problem instances (solved and unsolved) *RRHS* outperforms both *Hybrid* and *Oracle* for those problem instances that not solved by one of the solvers. The solution quality reported by *RRHS* is consistently the best over the cumulative fraction of problems solved.

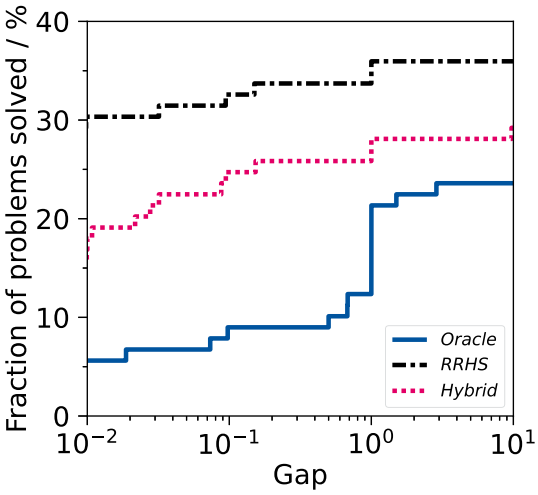


Fig. 5 Cumulative fraction of problem instances that are not solved by *at least one* of *RRHS*, *Hybrid*, and *Oracle* over the remaining objective value gap Gap (as defined in (6)). Solvers with tuned parameters, c.f., Table 7

These results reinforce the perception that *RRHS* with tuned parameters is the most effective solver.

From the sensitivity perspective (ii), similar to the results over all problem instances (solved and unsolved), c.f., Figure 2b, *RRHS* solved the most problems with the prescribed optimality tolerance of Gap $\leq 1 \times 10^{-2}$. We see that the number of solved problems and the results for large CPU time performance ratios in Figure 2 are not sensitive to an increase in the optimality tolerance. All solvers seem to behave similarly concerning the ability to solve an instance within the given time limit with looser termination tolerances. Assuming the problem instance is solved, we usually observe that looser termination tolerances generally facilitate faster convergence. Therefore, users should usually critically assess the necessity of (strict) tolerances. However, it appears that loosening termination tolerances does not significantly enhance solvability.

6.2.5 Performance result of *Minmax*

Figure 6 shows the performance of *Minmax* on the library of MINMAX test problems. We refrained from a comparison to SIP solvers using the reformulation given in (MINMAX-REF) as the specialized algorithm is preferable: The specialized algorithm provides the best upper-bound for each candidate point of the upper-level variables $\bar{\boldsymbol{x}}$ and the problems (RES), (ORA), and (UBP); used in the upper-bounding of the respective solvers; will not produce different candidate points (except when there are multiple possible global solutions, in which case better performance would only be due to randomness). *Minmax* was able to solve 79 % of the problem instances within 20 min. The remaining 21 % of problem instances were not solved due to subsolver errors, c.f., Table 8.

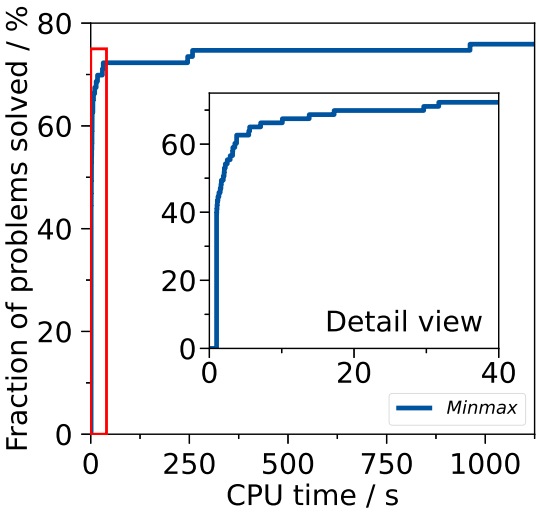


Fig. 6 CPU time performance profiles of *Minmax*. Most problems are solved within 40 s

6.3 Performance results of GSIP solvers

We conducted a similar parameter study for the GSIP solvers as for the SIP solvers. The problem instances of the GSIP benchmark study were solved using *GSIP-RRHS* and the corresponding derived SIPs, as described in Sect. 3.2, with the SIP solvers *Oracle*, *RRHS*, and *Hybrid*. We varied the parameters $\varepsilon^{\mathrm{r},0} = \{0.1, 1, 5\}$, $\varepsilon^{\mathrm{red}} = \{1.2, 2, 5\}$, $\alpha^0 = \{0.25, 0.5, 0.95\}$, and $\alpha^{\mathrm{red}} = \{1.2, 2, 5\}$ of the algorithms *GSIP-RRHS*, *RRHS*, and *Hybrid*, respectively. Figure 7 clearly shows that *GSIP-RRHS* performs best for many problems for tuned parameters (for CPU time, subproblem number, and subproblem performance ratio profiles see Figures S8 and S9 in Section S4.2 of the Supporting Information). One possible reason is that the SIP solvers solve the derived (LLP) with (GSIP-REF), while *GSIP-RRHS* solves the (GSIP-LLP) and (GSIP-AUX). The derived (LLP) is more expensive than (GSIP-LLP) and (GSIP-AUX) (c.f., Figure S10 in Section S4.2 of the Supporting Information). This is probably due to the non-smooth min in (GSIP-REF), which might lead to numerical disadvantages in the employed subsolver. However, it remains to be seen whether an improved opinionated subproblem formulation of (LLP) could alter these findings. We also want to point out that although *GSIP-RRHS* mostly outperforms the other solvers, it has more problems where it does not converge. An analysis analogous to the one in Sect. 6.2.4 shows that this is not changed significantly by increasing the optimality tolerance. This is connected to the issue raised by [54], which we try to address by reducing α iteratively whenever a GSIP-LLP-Slater point is not attained by (GSIP-AUX), c.f., Sects. 4.2.1 and 3.2. However, numerical issues connected to identifying GSIP-LLP-Slater points persist. For a detailed listing of the reasons why some of the problem instances were not solved, refer to Table 8.

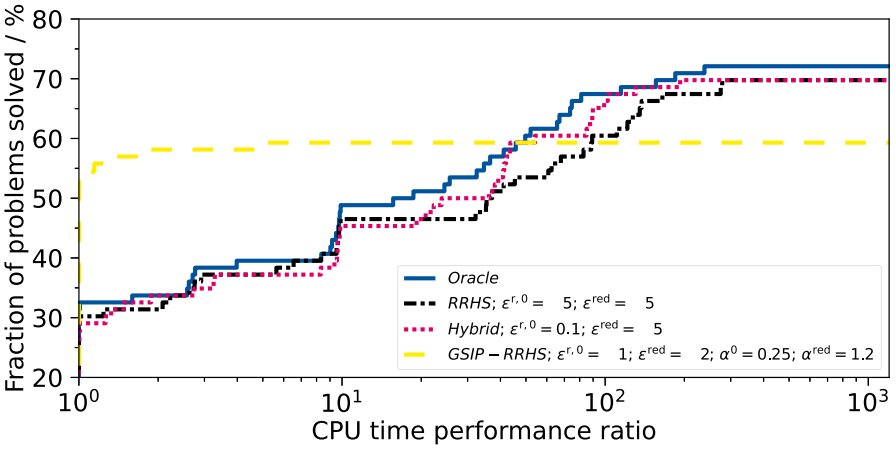


Fig. 7 CPU time performance ratio profiles of GSIP parameter study for *GSIP-RRHS*, *Oracle*, *RRHS*, and *Hybrid*, using tuned parameters according to Table 7. *GSIP-RRHS* is the fastest, most often, but solves fewer problems within the time limit

6.4 Performance results of BLP solvers

We conducted a similar parameter tuning study for the BLP solvers. For *BLP-Box*, we varied $d^0 = \{0.1, 0.5, 1\}$ and $d^{\text{red}} = \{0.1, 0.5, 0.9\}$. *BLP-noBox* performs best, c.f., Figure 8 (for CPU time, subproblem number, and subproblem performance ratio profiles see Figures S11 and S12 in Section S4.3 of the Supporting Information). Note that a large part of *BLP-Box*'s performance disadvantage is attributed to subsolver errors, c.f., Table 8. The *best* parameters for *BLP-Box* are $d^0 = 1$ and $d^{\text{red}} = 0.9$, while the influence of d^{red} is less significant. We conclude that the approach of choosing $\tilde{\mathscr{X}}$ of [28] is superior to [90].

For problem instances solved by none of the solvers with their respective parameters, the solvers did not provide meaningful UBD for a vast majority ($> 90\,\%$) of problem instances. In contrast to the analysis in Sect. 6.2.4, the fraction of unsolved problems (termination tolerance not met within the time limit) increased significantly by around $10\,\%$ for larger gaps (around 0.5) for *BLP-Box*.

6.5 Testing of two example scientific hypotheses

The extensive library of test problems, in combination with libDIPS, offers the advantage of quickly testing scientific hypotheses for the algorithms. The following sections briefly describe two hypotheses designed to accelerate convergence. Both were implemented in libDIPS and tested on the previously introduced library of test problems.

6.5.1 Introduction of a guard for the upper-bounding procedure in *RRHS*

RRHS, with tuned parameters ($\varepsilon^{\text{r},0} = 0.1$, $\varepsilon^{\text{red}} = 10$), spends about $24\,\%$ of the total CPU time for the upper-bounding procedure over all solved problem instances

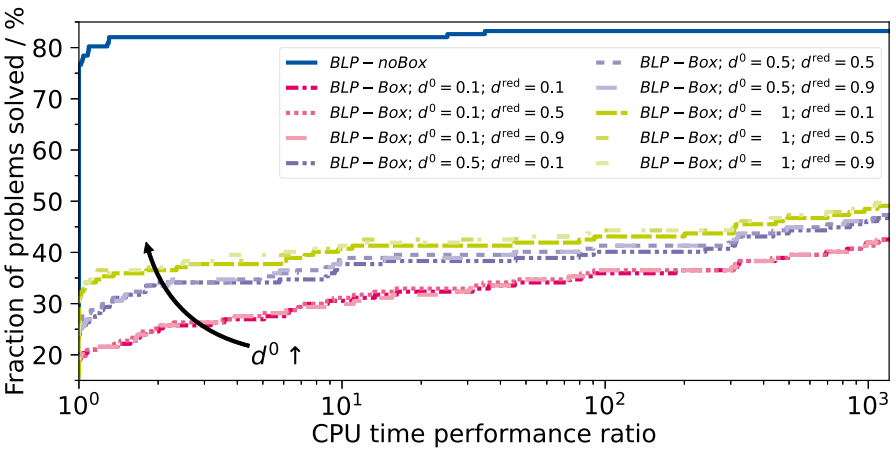


Fig. 8 CPU time performance ratio profiles of BLP parameter study for *BLP-noBox* and *BLP-Box* with $d^0 = \{0.1, 0.5, 1\}$ and $d^{\text{red}} = \{0.1, 0.5, 0.9\}$. *BLP-noBox* outperforms *BLP-Box* for all tested parameter values. Lines with the same color correspond to the same value of the parameter d^{red}. Changes in d^{red} appear to have little effect, while increasing d^0 improves performance

(c.f, Figure S3 in Section S4.1.1 of the Supporting Information). Recall that (UBP) of *RRHS* is generally neither a relaxation nor a restriction. If ε^{r} is a small value, the discretization must be "very" fine for the upper-bounding procedure to yield an SIP-feasible point solution and thus a "good" upper bound. Hence, if $\varepsilon^{\text{r},0}$ is a small value and we start with an empty discretization $\mathscr{Y}^{\text{UBP}} = \emptyset$, the first few upper-bounding iterations will most likely be unsuccessful. In an attempt to further reduce computation time, we added a guard such that the upper-bounding procedure of *RRHS* is skipped until the lower-bounding procedure produces ε^{a}-SIP-feasible points with $\varepsilon^{\text{a}} > 0$. Here, we choose $\varepsilon^{\text{a}} = 0.01$ and refer to this introduced guard parameter as $g^{\text{UBP}} = 0.01$. Whenever the upper-bounding procedure is skipped, the discretization of the $\mathscr{Y}^{\text{UBP}}$ is populated with the discretization point used in the lower-bounding procedure. Hence, the upper-bounding procedure is skipped until the discretization of $\mathscr{Y}^{\text{UBP}}$ has reached a sufficient density (in combination with ε^{r}). For a pseudocode of the described adapted algorithm, please refer to Algorithm 1 in Sect. C. In our experiments, this heuristic benefits problems where the solver *RRHS* was already competitive with a small CPU time performance ratio and does not negatively affect performance on the other problems. Indeed, in Figure 9, *RRHS* with the heuristic is the fastest solver for most problems, but the number of problems solved within the time limit does not change significantly. For subproblem number and subproblem ratio performance profiles, see Figure S13 in Section S4.4 of the Supporting Information. Note that this heuristic does not impede the convergence guarantees.

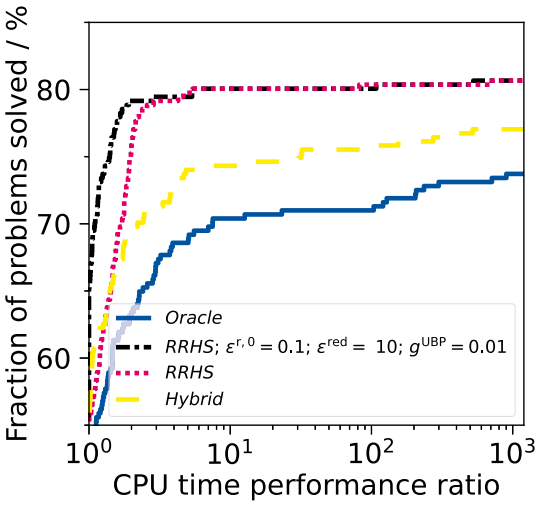


Fig. 9 CPU time performance ratio profiles for the introduced guard parameter as $g^{\text{UBP}} = 0.01$ in *RRHS*. Comparison with tuned parameters for the respective solvers. For small CPU time performance ratios, the guard significantly improves performance

6.5.2 Bracketing of the objective function by current upper and lower bound in *RRHS*

We test whether additional bracketing of the objective function speeds up convergence, i.e., we added to (LBP) and (UBP) the constraint

$$LBD \leq f(x) \leq UBD, \tag{7}$$

where LBD and UBD are the current upper and lower bounds. The idea is that bracketing of the objective function makes it easier for the used subsolvers to bound the solution. Additionally, if ε^{r} is "too" big but (UBP) is not directly infeasible, it is more likely that through the added bracketing constraints, (UBP) will become infeasible. Whenever (UBP) is infeasible, ε^{r} is deemed "too" large; it will be reduced, and the subsequent (unnecessary) solution of (LLP) is skipped.

We test our hypothesis on *RRHS* with tuned parameters, c.f, Table 7. We excluded the problems from [20] that were already shown in previous experiments to not converge within the time limit. In these problem instances, the upper-level problem is transformed from a QP to a QCQP by imposing constraints on the quadratic objective, potentially making the problem instances even more challenging.

The results shown in Figure 10 indicate that the additional bracketing of the objective function can have a considerable positive impact on a given problem instance. However, we do not see an overall improvement across the library of test problems. The bracketing of the objective function does not reduce the number of subproblems solved (for subproblem number and performance ratio profiles, see Figure S14 in Section S4.4 of the Supporting Information). In summary, bracketing the objective function is not a promising approach for reducing computation time and the number of subproblems to be solved.

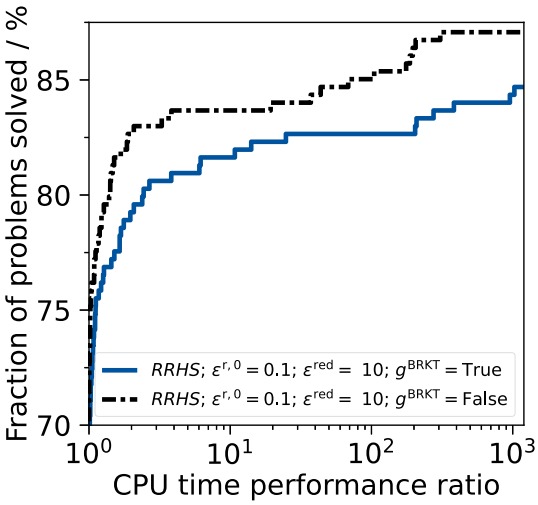


Fig. 10 CPU time performance ratio profiles comparing the approach using bracketing (g^{BRKT}=True) against the default of not using it (g^{BRKT}=False). Comparison with tuned parameters of *RRHS*. Bracketing does not improve performance. Problems from [20] are excluded

7 Conclusion and outlook

We presented libDIPS, an open-source software for adaptive discretization-based algorithms for (G)SIPs, BLPs, ESIPs, and MINMAXes, which provides solvers based on the algorithms proposed and further developed by [15, 28–30, 90, 92, 131]. We optimized the algorithmic parameters of the solvers and compared these "tuned" solvers on an extensive library of test problems comprising over 600 problem instances, which unified eight existing test sets using MAiNGO as a subsolver. We found that for SIPs and BLPs, the simpler algorithms outperform the more advanced ones, while for GSIPs, the more advanced tailored algorithm outperforms the simpler ones.

The implemented solvers, which all belong to the class of adaptive discretization-based algorithms, highly rely on the used subsolver. If the used subsolver performs poorly, the algorithm's performance is directly negatively affected. Therefore, it is recommended to try different subsolvers, as some subsolvers have strengths and weaknesses depending on the problem class (i.e., LP, NLP,...) and the problem formulation. Implicitly, subproblems with linear constraints and quadratic objectives are already handled by a specialized solver in our tests, as MAiNGO utilizes CPLEX in these cases. However, other problem classes or characteristics, such as convexity, could be exploited for faster and more robust optimization. Indeed, in our tests, we encountered a significant number of numerical difficulties in solving the subproblems. This is partially caused by the fact that with each level, stronger tolerances must be enforced.

As a result, the user should continually and critically evaluate their decision on the chosen feasibility tolerances, as they may be overly restrictive compared to the model accuracy. Restrictive feasibility tolerances can significantly influence the solution time. Additionally, they should explore whether a solution approach, which guarantees a feasible point upon termination, is necessary or if an ε^{a}-feasible point (with $\varepsilon^{\text{a}} > 0$) is sufficient, as generating a feasible point is more expensive.

The idea of trying different solvers also applies to the investigated adaptive discretization-based solvers. *Oracle* has excellent performance for some problems, while it struggles with other problems. Hence, it is likely beneficial to start with *Oracle* and try a different solver if *Oracle* fails to solve the problem instance in a reasonable time.

The presented library of test problems includes only a few problems with integers and a large portion of test problems in the library are solved quickly, several problems are unsolved, and only a small number of problems are solved within the time limit but take more than a minute to solve for the tested solvers. Due to the limiting number of integer problems, it remains an open research question whether the considered adaptive discretization-based algorithms and state-of-the-art algorithms for the (pure) integer case are computationally competitive, and if they are numerically as stable as the tailored approaches. This is especially the case for the BLP case, where only one of the solver currently supports integer variables. Concerning the solution times, it appears that many of the problems constructed as a challenge in previous publications are fast to solve after subsequent hardware and algorithmic improvements. Meanwhile, others are hardly tractable with the investigated approaches (in combination with the used subsolver MAiNGO). Therefore, it is likely necessary to use expert knowledge of the problem to facilitate its solution (if the problem is hard). Note that libDIPS offers the possibility of formulating the subproblems independently in an opinionated way. It remains to be seen whether the unsolved problems are inherently difficult to solve or whether other (non-)adaptive discretization-based solution approaches would excel at solving them. Classifying the problems into difficulty classes would also be a necessary step in refining our test collection to a proper benchmark. Ideally, a standard file format would greatly benefit such a community benchmark.

Similar to the potential gaps identified in the difficulty level, the categorization of the instances in the library of test problems revealed that problems of several problem categories, e.g., MINMAXes, with an upper level being QCQP, are underrepresented. This also holds for problems with discrete variables and application-based test problem instances. Hence, future work should consider more test problem instances, especially application-based ones. To facilitate easy extension and reuse of the library of test problems, interested readers may suggest additional problem instances via the Git issue system in the libDIPS repository.

Apart from further expanding and improving the library of test problems, we believe there are three main promising directions for improvement: initialization strategies for the discretization, subproblem formulations and adaptations, and algorithm adaptations, including heuristics and extensions. One could apply an (advanced) initialization of the discretized sets instead of starting with empty ones, e.g., by initializing the sets with KKT points of the lower-level problem or with edge points of semi-infinite sets. To reduce the number of iterations, the subproblems could be adapted by using better-suited formulations for the used subsolver, adding KKT conditions to the upper-level problems, or considering higher-order cuts, as in [26]. Note that the first proposed adaptation is already possible in libDIPS. However, this will likely make the subproblems more expensive to solve. Last but not least, algorithms can be adapted and additional ones can be implemented, e.g., the interval-based method of [13, 14]. For SIPs, promising ideas for algorithm adaptations include employing the solver *B&F*, and then, after a given feasibility tolerance is met, using (RES) to search a feasible

point that meets the given optimality tolerance; and testing the impact of adapting *Hybrid* to a local solution of the (RES) subproblem. For GSIPs, hybridization in the generation of the discretization points of *GSIP-RRHS* is promising. *GSIP-RRHS* outperforms the other solvers. However, if *GSIP-RRHS* fails to converge, i.e., when no GSIP-LLP-Slater point can be produced through (GSIP-AUX), using the (LLP) with (GSIP-REF) might increase the number of solved problems. Alternatively, one could solve (LLP) with (GSIP-REF) every n iterations and populate the discretization with its solution. The impact of all these adaptations can be easily evaluated using libDIPS and the presented library of test problems.

Appendix A Details to ESIP

ESIPs allow the modeling of an existence constraint instead of a semi-infinite constraint and can be considered as SIPs with another hierarchical problem embedded. In this sense, they can be considered a three-level optimization problem with a third level of variables. Thus, in ESIPs we consider the variables $\boldsymbol{x}$ as the upper-level, $\boldsymbol{y}$ as the medial-level, and $\boldsymbol{z}$ as the lower-level variables. The lower-level variables $\boldsymbol{z}$ consist of $n_{\mathrm{z,c}}$ continuous and $n_{\mathrm{z,i}}$ integer variables, with the host set defined as $\bar{\mathscr{Z}} := \left[\boldsymbol{z}^{\mathrm{lb}}, \boldsymbol{z}^{\mathrm{ub}}\right] \cap (\mathbb{R}^{n_{\mathrm{z,c}}} \times \mathbb{Z}^{n_{\mathrm{z,i}}})$. The corresponding host sets of $\boldsymbol{x}$ and $\boldsymbol{y}$ are defined as before. The ESIP reads

$$
\begin{aligned}
&\min_{\boldsymbol{x} \in \mathscr{X}} \; f(\boldsymbol{x}) \\
&\quad \text{s.t. } \forall\, \boldsymbol{y} \in \mathscr{Y} \left[\exists \boldsymbol{z} \in \mathscr{Z} : \boldsymbol{g}^{\mathrm{u}}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) \leq \boldsymbol{0}\right] \\
&\text{with } \mathscr{X} := \left\{\boldsymbol{x} \in \bar{\mathscr{X}} : \boldsymbol{v}^{\mathrm{iu}}(\boldsymbol{x}) \leq \boldsymbol{0},\ \boldsymbol{v}^{\mathrm{eu}}(\boldsymbol{x}) = \boldsymbol{0}\right\} \\
&\qquad \mathscr{Y} := \left\{\boldsymbol{y} \in \bar{\mathscr{Y}} : \boldsymbol{v}^{\mathrm{il}}(\boldsymbol{y}) \leq \boldsymbol{0},\ \boldsymbol{v}^{\mathrm{el}}(\boldsymbol{y}) = \boldsymbol{0}\right\} \\
&\qquad \mathscr{Z} := \left\{\boldsymbol{z} \in \bar{\mathscr{Z}} : \boldsymbol{v}^{\mathrm{ie}}(\boldsymbol{z}) \leq \boldsymbol{0},\ \boldsymbol{v}^{\mathrm{ee}}(\boldsymbol{z}) = \boldsymbol{0}\right\};
\end{aligned}
\tag{ESIP}
$$

with $\boldsymbol{g}^{\mathrm{u}} : \mathbb{R}^{n_{\mathrm{x}}} \times \mathbb{R}^{n_{\mathrm{y}}} \times \mathbb{R}^{n_{\mathrm{z}}} \to \mathbb{R}^{n_{\mathrm{gu}}}$, $\boldsymbol{v}^{\mathrm{ie}} : \mathbb{R}^{n_{\mathrm{z}}} \to \mathbb{R}^{n_{\mathrm{vie}}}$, $\boldsymbol{v}^{\mathrm{ee}} : \mathbb{R}^{n_{\mathrm{z}}} \to \mathbb{R}^{n_{\mathrm{vee}}}$, and all other functions and sets as defined for (SIP). Note that in the case of ESIPs, we write the logical quantifiers in prefix notation because the order of the quantifiers $\forall$ and $\exists$ is essential.

Generalizations of (ESIP) to cases where the lower-level feasible set $\mathscr{Y}$ depends on the upper-level variables $\boldsymbol{x}$, or the feasible set of existence-constrained variables $\mathscr{Z}$ depends on the lower-level variables $\boldsymbol{y}$ or the upper-level variables $\boldsymbol{x}$ are straightforward, but not yet implemented in libDIPS. See [30] for these extensions.

A.1 Approaches for ESIPs

The algorithmic extension to ESIPs, proposed in [30], can similarly be understood as a direct extension of the ideas for SIP. Instead of the single-level optimization problem

(LLP), the following max-min problem

$$\max_{\boldsymbol{y}\in\mathscr{Y}} \min_{\boldsymbol{z}\in\mathscr{Z}} \max_{j\in\{1\dots n_{\mathrm{gu}}\}} g_j^{\mathrm{u}}(\bar{\boldsymbol{x}}, \boldsymbol{y}, \boldsymbol{z}) \quad \text{with } \mathscr{Y} \coloneqq \left\{\boldsymbol{y} \in \bar{\mathscr{Y}} : \boldsymbol{v}^{\mathrm{il}}(\boldsymbol{y}) \leq \boldsymbol{0},\ \boldsymbol{v}^{\mathrm{el}}(\boldsymbol{y}) = \boldsymbol{0}\right\} \tag{ESIP-MINMAX}$$

takes the place of the lower-level problem. A lower bound is obtained by solving the subproblem

$$\min_{\boldsymbol{x}\in\mathscr{X}, \boldsymbol{z}^1\in\mathscr{Z}\dots\boldsymbol{z}^{|\mathscr{Y}^{LBP}|}\in\mathscr{Z}} f(\boldsymbol{x}) \quad \text{s.t. } \boldsymbol{g}^{\mathrm{u}}\left(\boldsymbol{x}, \boldsymbol{y}^k, \boldsymbol{z}^k\right) \leq \boldsymbol{0},\ \forall\, k \in \left\{1 \dots \left|\mathscr{Y}^{\mathrm{LBP}}\right|\right\} \quad \text{with } \mathscr{X} \coloneqq \left\{\boldsymbol{x} \in \bar{\mathscr{X}} : \boldsymbol{v}^{\mathrm{iu}}(\boldsymbol{x}) \leq \boldsymbol{0},\ \boldsymbol{v}^{\mathrm{eu}}(\boldsymbol{x}) = \boldsymbol{0}\right\}, \tag{ESIP-LBP}$$

where the key difference to (LBP) is the addition of a new entry for the existence variables $\boldsymbol{z}$ for each discretization point in $\mathscr{Y}^{\mathrm{LBP}}$.

Appendix B Further subproblem formulations

B.1 SIP Solvers

(ORA) of *Oracle* can be reformulated as

$$\min_{\boldsymbol{x}\in\mathscr{X}, \nu} \nu \quad \text{s.t. } f(\boldsymbol{x}) - f^{\mathrm{t}} \leq \nu,\ \boldsymbol{g}^{\mathrm{u}}\left(\boldsymbol{x}, \boldsymbol{y}^k\right) \leq \nu \cdot \boldsymbol{1},\ \forall\, \boldsymbol{y}^k \in \mathscr{Y}^{\mathrm{ORA}} \quad \text{with } \mathscr{X} \coloneqq \left\{\boldsymbol{x} \in \bar{\mathscr{X}} : \boldsymbol{v}^{\mathrm{iu}}(\boldsymbol{x}) \leq \boldsymbol{0},\ \boldsymbol{v}^{\mathrm{eu}}(\boldsymbol{x}) = \boldsymbol{0}\right\}. \tag{ORA-REF}$$

B.2 MINMAX Solver

The upper-level problem of *Minmax* reads

$$\min_{\boldsymbol{x}\in\mathscr{X}, t} t \quad \text{s.t. } f\left(\boldsymbol{x}, \boldsymbol{y}^k\right) - t \leq 0\, \forall\, \boldsymbol{y}^k \in \mathscr{Y}^{\mathrm{LBP}} \quad \text{with } \mathscr{X} \coloneqq \left\{\boldsymbol{x} \in \bar{\mathscr{X}} : \boldsymbol{v}^{\mathrm{iu}}(\boldsymbol{x}) \leq \boldsymbol{0},\ \boldsymbol{v}^{\mathrm{eu}}(\boldsymbol{x}) = \boldsymbol{0}\right\}. \tag{MINMAX-LBP}$$

Note that we have introduced the auxiliary variable t.

B.3 ESIP Solver

The lower-level problem of (ESIP-MINMAX) of *B&F* for fixed $\bar{\boldsymbol{y}}$ and $\bar{\boldsymbol{x}}$ reads

$$\min_{z \in \mathscr{Z}} \max_{j \in \{1 \ldots n_{\mathrm{gu}}\}} g_j^{\mathrm{u}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}, \boldsymbol{z}) \tag{ESIP-LLP}$$

The upper-level problem of (ESIP-MINMAX) for fixed $\bar{\boldsymbol{x}}$ and a finite set of discretization points $\mathscr{Z}^{MLP}$ is formulated as

$$\begin{aligned} &\max_{\boldsymbol{y} \in \mathscr{Y}} \min_{\boldsymbol{z} \in \mathscr{Z}^{\mathrm{MLP}}} \max_{j \in \{1 \ldots n_{\mathrm{gu}}\}} g_j^{\mathrm{u}}(\bar{\boldsymbol{x}}, \boldsymbol{y}, \boldsymbol{z}) \\ &\text{with } \mathscr{Y} := \left\{ \boldsymbol{y} \in \bar{\mathscr{Y}} : \boldsymbol{v}^{\mathrm{il}}(\boldsymbol{y}) \leq \boldsymbol{0},\ \boldsymbol{v}^{\mathrm{el}}(\boldsymbol{y}) = \boldsymbol{0} \right\}. \end{aligned} \tag{ESIP-MLP}$$

B.4 BLP Solvers

The auxiliary problem of *BLP-noBox* and *BLP-Box* is formulated as

$$\begin{aligned} &\min_{\boldsymbol{y} \in \mathscr{Y}(\bar{\boldsymbol{x}}), u \in \mathbb{R}} \quad u \\ &\quad \text{s.t. } h(\bar{\boldsymbol{x}}, \boldsymbol{y}) \leq h^*(\bar{\boldsymbol{x}}) + \varepsilon^{\mathrm{AUX}} \\ &\qquad \boldsymbol{g}^l(\bar{\boldsymbol{x}}, \boldsymbol{y}) \leq u \cdot \boldsymbol{1} \\ &\text{with } \mathscr{Y}(\boldsymbol{x}) := \left\{ \boldsymbol{y} \in \bar{\mathscr{Y}} : \boldsymbol{g}^{\mathrm{l}}(\boldsymbol{x}, \boldsymbol{y}) \leq \boldsymbol{0},\ \boldsymbol{v}^{\mathrm{il}}(\boldsymbol{y}) \leq \boldsymbol{0},\ \boldsymbol{v}^{\mathrm{el}}(\boldsymbol{y}) = \boldsymbol{0} \right\}. \end{aligned} \tag{BLP-AUX}$$

The auxiliary problem for *BLP-Box* for deciding whether or not a given box $\mathscr{X}_{\mathrm{box}}$ for the upper-level variables is small enough such that the given discretization point $\bar{\boldsymbol{y}}$ stays feasible in the lower level is given by

$$\min_{\boldsymbol{x} \in \mathscr{X}_{\mathrm{box}}} - \max_{j \in \{1 \ldots n_{\mathrm{gl}}\}} \boldsymbol{g}^l(\boldsymbol{x}, \bar{\boldsymbol{y}}). \tag{BLP-AUX-V}$$

Note that [90] uses interval analysis to determine if $\mathscr{X}_{\mathrm{box}}$ is valid.

Appendix C Details on UBP-Guard

The pseudocode in Algorithm 1 illustrates the experimental change described in Sect. 6.5. It also includes one of the changes mentioned in Sect. 4.2.1, namely the recovery step (see Sect. S2 of the Supporting Information). Other details, such as consideration of optimality tolerances, are omitted for brevity.

Algorithm 1: Simplified pseudocode of *RRHS* with guard g^{UBP} for the upper-bounding procedure and recovery step.

$LBD \leftarrow -\infty, UBD \leftarrow \infty, \mathscr{Y}^{\mathrm{LBP}} \leftarrow \emptyset, \mathscr{Y}^{\mathrm{UBP}} \leftarrow \emptyset, \varepsilon^{\mathrm{r}} \leftarrow \varepsilon^{\mathrm{r},0}, NEXT \leftarrow LBP, i \leftarrow 0$;
while ($i <$ `maxiter`) ***and*** LBD *and* UBD ***not*** *sufficiently close* **do**
  switch $NEXT$ **do**
    case *LBP* **do**
      Solve (LBP) to obtain f^{LBP} and $\bar{\boldsymbol{x}}$;
      if *Infeasible* **then return** Infeasible ;
      $LBD \leftarrow f^{\mathrm{LBP}}$;
      Solve (LLP) given $\bar{\boldsymbol{x}}$ to obtain g^{LLP} and $\bar{\boldsymbol{y}}$;
      if *Infeasible* ***or*** $g^{LLP} \leq 0$ **then**
        $UBD \leftarrow f^{\mathrm{LBP}}$;
        return GLOBAL;
      end
      $\mathscr{Y}^{\mathrm{LBP}} \leftarrow \mathscr{Y}^{\mathrm{LBP}} \cup \{\bar{\boldsymbol{y}}\}$;
      if $g^{LLP} < g^{UBP}$ ***or*** $g^{UBP} \leq 0$ **then**
        $NEXT \leftarrow UBP$;
      else
        $\mathscr{Y}^{\mathrm{UBP}} \leftarrow \mathscr{Y}^{\mathrm{UBP}} \cup \{\bar{\boldsymbol{y}}\}, NEXT \leftarrow LBP$;
      end
    end
    case *UBP* **do**
      Solve (UBP) to obtain f^{UBP} and $\bar{\boldsymbol{x}}$;
      if *Infeasible* **then**
        $\varepsilon^{\mathrm{r}} \leftarrow \varepsilon^{\mathrm{r}}/\varepsilon^{\mathrm{red}}$;
        if $\varepsilon^{r} < \varepsilon^{r,min}$ **then** $NEXT \leftarrow RECOVER$;
      else
        Solve (LLP) given $\bar{\boldsymbol{x}}$ to obtain g^{LLP} and $\bar{\boldsymbol{y}}$;
        if *Infeasible* ***or*** $g^{LLP} \leq 0$ **then**
          $UBD \leftarrow f^{\mathrm{UBP}}, \varepsilon^{\mathrm{r}} \leftarrow \varepsilon^{\mathrm{r}}/\varepsilon^{\mathrm{red}}$;
          if $\varepsilon^{r} < \varepsilon^{r,min}$ **then** $NEXT \leftarrow RECOVER$;
        else
          $\mathscr{Y}^{\mathrm{UBP}} \leftarrow \mathscr{Y}^{\mathrm{UBP}} \cup \{\bar{\boldsymbol{y}}\}$;
        end
      end
    end
    case *RECOVER* **do**
      Solve (UBP) with $\varepsilon^{\mathrm{r}} = 0$ and obtain f^{REC} and $\bar{\boldsymbol{x}}$;
      if *Infeasible* **then return** Infeasible ;
      Solve (LLP) given $\bar{\boldsymbol{x}}$ to obtain g^{LLP} and $\bar{\boldsymbol{y}}$;
      if *Infeasible* ***or*** $g^{LLP} \leq 0$ **then**
        $LBD \leftarrow f^{\mathrm{REC}}, UBD \leftarrow f^{\mathrm{REC}}$;
        return GLOBAL;
      end
      return ASSUMPTION_VIOLATION_LIKELY;
    end
  end
end

Appendix D Definitions

D.1 (ε^{a})-Feasibility

Definition 1 *(ε^{a}-SIP-Feasible Point)* A point $\bar{\boldsymbol{x}} \in \mathscr{X}$ is ε^{a}-SIP-feasible in (SIP) if

$$\boldsymbol{g}^{\mathrm{u}}(\bar{\boldsymbol{x}}, \boldsymbol{y}) \leq \varepsilon^{\mathrm{a}} \cdot \mathbf{1}, \ \forall\, \boldsymbol{y} \in \mathscr{Y},$$

with $\varepsilon^{\mathrm{a}} \geq 0$.

Definition 2 *(ε^{a}-GSIP-Feasible Point)* A point $\bar{\boldsymbol{x}} \in \mathscr{X}$ is ε^{a}-GSIP-feasible in (GSIP) if

$$\min \left\{ g_i^{\mathrm{u}}(\bar{\boldsymbol{x}}, \boldsymbol{y}), \min_{j \in \{1 \ldots n_{\mathrm{gl}}\}} - g_j^{\mathrm{l}}(\bar{\boldsymbol{x}}, \boldsymbol{y}) \right\} \leq \varepsilon^{\mathrm{a}}, \ \forall\, i \in \left\{1 \ldots n_{\mathrm{gu}}\right\}, \ \forall\, \boldsymbol{y} \in \hat{\mathscr{Y}},$$

with $\varepsilon^{\mathrm{a}} \geq 0$ and $\hat{\mathscr{Y}}$ as defined in (3) in Sect. 3.2.

Definition 3 *((G)SIP-Feasible Point)* A point $\bar{\boldsymbol{x}} \in \mathscr{X}$ is called (G)SIP-feasible in (SIP) ((GSIP)) if it fulfills Definition 1 (Definition 2) with $\varepsilon^{\mathrm{a}} = 0$.

Definition 4 *(ε^{a}-BLP-Feasible Point)* A pair $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$ with $\bar{\boldsymbol{x}} \in \mathscr{X}$ and $\bar{\boldsymbol{y}} \in \mathscr{Y}(\bar{\boldsymbol{x}})$ is called ε^{a}-BLP-feasible in (BLP) if $\boldsymbol{g}^{\mathrm{u}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \leq \mathbf{0}$ and $\bar{\boldsymbol{y}}$ is ε^{a}-optimal in (BLP-LLP), see [90].

D.2 SIP-Slater and GSIP-LLP-slater point

Definition 5 *(SIP-Slater Point)* A point $\boldsymbol{x}^{\mathrm{S}} \in \mathscr{X}$ is called an SIP-Slater point in (SIP) if

$$\boldsymbol{g}^{\mathrm{u}}\left(\boldsymbol{x}^{\mathrm{S}}, \boldsymbol{y}\right) < \mathbf{0}, \ \forall\, \boldsymbol{y} \in \mathscr{Y}.$$

Under compactness of $\mathscr{Y}$ and continuity of $\boldsymbol{g}$ if $\boldsymbol{x}^{\mathrm{S}}$ is an SIP-Slater point, there exists $\varepsilon^{\mathrm{S}} > 0$, such that

$$\boldsymbol{g}^{\mathrm{u}}\left(\boldsymbol{x}^{\mathrm{S}}, \boldsymbol{y}\right) \leq -\varepsilon^{\mathrm{S}} \cdot \mathbf{1}, \ \forall\, \boldsymbol{y} \in \mathscr{Y},$$

c.f., Definition A.1 in [89].

Definition 6 *(GSIP-LLP-Slater Point)* A point $\boldsymbol{y}^{\mathrm{S}} \in \hat{\mathscr{Y}}$ (see (3) in Sect. 3.2) is called a GSIP-LLP-Slater point at $\bar{\boldsymbol{x}}$ in (GSIP) if

$$\boldsymbol{g}^{\mathrm{l}}\left(\bar{\boldsymbol{x}}, \boldsymbol{y}^{\mathrm{S}}\right) \leq -\varepsilon^{\mathrm{S}} \cdot \mathbf{1},$$

with some $\varepsilon^{\mathrm{S}} > 0$, c.f., Appendix B.2 in [92].

D.3 ε^{f}-optimal (G)SIP-slater point

Definition 7 *(ε^{f}-optimal SIP-Slater Point)* A point $\boldsymbol{x}^{\mathrm{S}} \in \mathscr{X}$ is called an ε^{f}-optimal SIP-Slater point in (SIP) if

$$f\left(\boldsymbol{x}^{\mathrm{S}}\right) \leq f^{*} + \varepsilon^{\mathrm{f}} \wedge \boldsymbol{g}^{\mathrm{u}}\left(\boldsymbol{x}^{\mathrm{S}}, \boldsymbol{y}\right) \leq -\varepsilon^{\mathrm{S}} \cdot \mathbf{1},\ \forall\, \boldsymbol{y} \in \mathscr{Y},$$

with $\varepsilon^{\mathrm{f}}, \varepsilon^{\mathrm{S}} > 0$, c.f., Lemma 2.4 in [89].

Definition 8 *(ε^{f}-optimal GSIP-Slater Point)* A point $\boldsymbol{x}^{\mathrm{S}} \in \mathscr{X}$ is called an ε^{f}-optimal GSIP-Slater point in (GSIP) if

$$f\left(\boldsymbol{x}^{\mathrm{S}}\right) \leq f^{*} + \varepsilon^{\mathrm{f}} \wedge \left[\boldsymbol{g}^{\mathrm{u}}\left(\boldsymbol{x}^{\mathrm{S}}, \boldsymbol{y}\right) \leq -\varepsilon^{\mathrm{S}} \cdot \mathbf{1} \vee \exists j : g_j^{\mathrm{l}}\left(\boldsymbol{x}^{\mathrm{S}}, \boldsymbol{y}\right) > \varepsilon^{\mathrm{S}}\right],\ \forall\, \boldsymbol{y} \in \hat{\mathscr{Y}},$$

with $\varepsilon^{\mathrm{f}}, \varepsilon^{\mathrm{S}} > 0$, and $\hat{\mathscr{Y}}$ as defined in (3) in Sect. 3.2, c.f., Assumption 3 in [92].

Appendix E Note on integer capabilities of the solvers

In the following, we argue that although some of the publications on which we base our implementation do not explicitly cover the integer case, the corresponding implementation handles discrete variables. The exception is *BLP-Box*, as we base our implementation on [90]. Although the extension in [88] explicitly covers the integer case, the required modifications have not yet been implemented.

[28] considers BLPs and GSIPs and [88] considers BLPs with mixed-integers. Both approaches provide a proof of finite convergence for their respective methods. Recall that the solver *BLP-noBox* is based on [28] and [88] is the extension to [90], which is implemented as *BLP-Box*.

The approach presented in [28] extends the methods of [15] (*B&F*), [89] (*RRHS*), and [92] (*GSIP-RRHS*). The convergence proof provided in [28] can be adapted to these earlier approaches to establish finite convergence. Similarly, although [131] (*Oracle*) and [29] (*Hybrid*) do not explicitly address integer variables, the algorithms are based on [15], and the same convergence argument applies.

To illustrate the underlying concept, we note that all approaches approximate the set $\mathscr{Y}$ with a finite discrete set. Similarly, there are only finitely many possible values for the discrete lower-variables within the host set. Thus, in many cases, it is straightforward to incorporate host sets $\bar{\mathscr{Y}}$ that restrict a subset of the variables to be discrete without affecting convergence guarantees. In fact, discretization of the discrete dimensions is inherently more effective, as the set of feasible values is already finite. Therefore, assuming finite convergence is established for fixed values of the discrete variables, the approximation will be exact with respect to the discrete variables after a finite number of additional iterations (a more detailed discussion, applicable to the other algorithms as well, can be found in [28, 88]). In the worst case, this may require complete enumeration of the discrete lower-level variables. However, in our experience with applications involving integer variables, only a few iterations are

typically required (see, e.g., [27, 150] and our computational results for SIPs with discrete variables in the upper and/or lower level, c.f., Figure S7 in Section S4.1.3 of the Supporting Information).

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/s12532-026-00308-3.

Acknowledgements We thank the anonymous reviewers for their constructive comments and valuable suggestions, which have led to significant improvements in this manuscript. We acknowledge the use of the GPT-3.5 language model developed by OpenAI as an implementation aid, i.e., as a code snippet generator, for creating shell scripts to automate the execution of the benchmark tests on the RWTH High Performance Computing cluster and for generating ideas for converting data formats to create the plots presented in this work. During the preparation of this work, the authors utilized Grammarly's writing assistant tool and GPT language models developed by OpenAI to enhance the readability and language of the manuscript by generating ideas for improving previously drafted text. After using this tool/service, the authors reviewed and edited the content as needed and take full responsibility for the content of the publication. Computations were performed with computing resources granted by RWTH Aachen University. Special thanks to Jan-Frederic Laub, Yiju Chen, Inken Michael, Finja Backhaus, Jan Eifert, Julia Jasovski, Jason Klein and My Pham for their help in collecting and implementing (G)SIPs. Further, we thank Adrian Lipow for the insightful and fruitful discussions.

Author Contributions All authors contributed to the study conception and design. Material preparation, data collection and analysis were performed by Aron Zingler and Daniel Jungen. Aron Zingler and Daniel Jungen contributed equally to this work. The first draft of the manuscript was written by Aron Zingler and Daniel Jungen and all authors commented on previous versions of the manuscript. Hatim Djelassi implemented the first version of the software. Aron Zingler and Daniel Jungen are continuing the development of the software under supervision of Alexander Mitsos. All authors read and approved the final manuscript.

Funding Open Access funding enabled and organized by Projekt DEAL. Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – Cluster of Excellence 2186 "The Fuel Science Center" – ID: 390919832. We gratefully acknowledge the financial support provided by Réseau de transport d'électricité (RTE, France) through the project "Hierarchical Optimization for Worst-case and Flexibility Analysis of Power Grids".

Data Availability The benchmark results generated and analysed during the current study are available at https://doi.org/10.18154/RWTH-2023-10589 in a json file format.

Declarations

Statements and Declarations A previous version of this work was published as a preprint [67], with parts presented at scientific conferences [66, 68] and included in the authors' dissertation [64, 149].

Competing Interests The authors have no relevant financial or non-financial interests to disclose.

References

1. Anderson, E.J., Lewis, A.S.: An extension of the simplex algorithm for semi-infinite linear programming. Math. Program. **44**(1–3), 247–269 (1989). https://doi.org/10.1007/BF01587092
2. Andreassen, D.O., Watson, G.A.: Linear Chebyshev approximation without Chebyshev sets. BIT Numer. Math. **16**(4), 349–362 (1976). https://doi.org/10.1007/BF01932717
3. Atkinson, A.C., Federov, V.V.: The design of experiments for discriminating between two rival models. Biometrika **62**(1), 57–70 (1975). https://doi.org/10.1093/biomet/62.1.57
4. Auslender, A., Ferrer, A., Goberna, M.A., López, M.A.: Comparative study of RPSALG algorithm for convex semi-infinite programming. Comput. Optim. Appl. **60**(1), 59–87 (2015). https://doi.org/10.1007/s10589-014-9667-7
5. Avraamidou, S., Pistikopoulos, E.N.: B-POP: Bi-level parametric optimization toolbox. Comput. Chem. Eng. **122**, 193–202 (2019). https://doi.org/10.1016/j.compchemeng.2018.07.007
6. Barragán, A., Camacho-Vallejo, J.F.: An exact algorithm based on the Kuhn-Tucker conditions for solving linear generalized semi-infinite programming problems. J. Math. **2022**, 1–14 (2022). https://doi.org/10.1155/2022/1765385
7. Barrodale, I., Delves, L.M., Mason, J.C.: Linear Chebyshev approximation of complex-valued functions. Math. Comput. **32**(143), 853 (1978). https://doi.org/10.2307/2006490
8. Beck, Y., Bienstock, D., Schmidt, M., Thürauf, J.: On a computationally ill-behaved bilevel problem with a continuous and nonconvex lower level. J. Optim. Theory Appl. **198**(1), 428–447 (2023). https://doi.org/10.1007/s10957-023-02238-9
9. Ben-Tal, A., Nemirovski, A.: Robust solutions of uncertain linear programs. Oper. Res. Lett. **25**(1), 1–13 (1999). https://doi.org/10.1016/S0167-6377(99)00016-4
10. Ben-Tal, A., Nemirovski, A.: Robust optimization - methodology and applications. Math. Program. **92**(3), 453–480 (2002). https://doi.org/10.1007/s101070100286
11. Betro, B.: An accelerated central cutting plane algorithm for linear semi-infinite programming. Math. Program. **101**(3), 479–495 (2004). https://doi.org/10.1007/s10107-003-0492-5
12. Beykal, B., Avraamidou, S., Pistikopoulos, I.P.E., Onel, M., Pistikopoulos, E.N.: DOMINO: Data-driven Optimization of bi-level Mixed-Integer NOnlinear Problems. J. Global Optim. **78**(1), 1–36 (2020). https://doi.org/10.1007/s10898-020-00890-3
13. Bhattacharjee, B., Green, W.H., Jr., Barton, P.I.: Interval methods for semi-infinite programs. Comput. Optim. Appl. **30**(1), 63–93 (2005). https://doi.org/10.1007/s10589-005-4556-8
14. Bhattacharjee, B., Lemonidis, P., Green, W.H., Jr., Barton, P.I.: Global solution of semi-infinite programs. Math. Program. **103**(2), 283–307 (2005). https://doi.org/10.1007/s10107-005-0583-6
15. Blankenship, J.W., Falk, J.E.: Infinitely constrained optimization problems. J. Optim. Theory Appl. **19**(2), 261–281 (1976). https://doi.org/10.1007/BF00934096
16. Bomze, I., Horländer, A., Schmidt, M.: Mixed-integer bilevel optimization with nonconvex quadratic lower-level problems: Complexity and a solution method. J. Global Optim. (2025). https://doi.org/10.1007/s10898-025-01522-4
17. Bongartz, D., Najman, J., Sass, S., Mitsos, A.: MAiNGO – McCormick-based Algorithm for mixed-integer Nonlinear Global Optimization (2018). http://www.avt.rwth-aachen.de/global/show_document.asp?id=aaaaaaaaabclahw. Accessed 2023/07/20
18. Bonnans, J.F., Shapiro, A.: Perturbation analysis of optimization problems. Springer Series in Operations Research. Springer, New York, USA (2000)
19. Cánovas, M.J., López, M.A., Parra, J., Todorov, M.I.: Stability and well-posedness in linear semi-infinite programming. SIAM Journal on Optimization: A Publication of the Society for Industrial and Applied Mathematics **10**(1), 82–98 (1999). https://doi.org/10.1137/S1052623497319869
20. Cerulli, M., Oustry, A., d'Ambrosio, C., Liberti, L.: Convergent algorithms for a class of convex semi-infinite programs. SIAM Journal on Optimization: A Publication of the Society for Industrial and Applied Mathematics **32**(4), 2493–2526 (2022). https://doi.org/10.1137/21M1431047
21. Chen, Z.: Optimality conditions of semi-infinite programming and generalized semi-infinite programming. Ph.D. thesis, The Hong Kong Polytechnic University, Hong Kong (2013)
22. Clark, P.A., Westerberg, A.W.: Bilevel programming for steady-state chemical process design - I. fundamentals and algorithms. Comput. Chem. Eng. **14**(1), 87–97 (1990). https://doi.org/10.1016/0098-1354(90)87007-C

23. Coope, I.D., Price, C.J.: Exact penalty function methods for nonlinear semi-infinite programming. In: Pardalos, P., Horst, R., Reemtsen, R., Rückmann, J.J. (eds.) Semi-Infinite Programming, Nonconvex Optimization and Its Applications, vol. 25, pp. 137–157. Springer, US, Boston, MA, USA (1998)
24. Curtis, A.R., Powell, M.J.D.: Necessary conditions for a minimax approximation. Comput. J. **8**(4), 358–361 (1966). https://doi.org/10.1093/comjnl/8.4.358
25. Diehl, M., Houska, B., Stein, O., Steuermann, P.: A lifting method for generalized semi-infinite programs based on lower level Wolfe duality. Comput. Optim. Appl. **54**(1), 189–210 (2013). https://doi.org/10.1007/s10589-012-9489-4
26. Djelassi, H.: Discretization-based algorithms for the global solution of hierarchical programs. Dissertation, RWTH Aachen University, Aachen (2020). https://doi.org/10.18154/RWTH-2020-09163
27. Djelassi, H., Fliscounakis, S., Mitsos, A., Panciatici, P.: Hierarchical programming for worst-case analysis of power grids. In: 2018 Power Systems Computation Conference (PSCC), pp. 1–7 (2018). https://doi.org/10.23919/PSCC.2018.8444136
28. Djelassi, H., Glass, M., Mitsos, A.: Discretization-based algorithms for generalized semi-infinite and bilevel programs with coupling equality constraints. J. Global Optim. **75**(2), 341–392 (2019). https://doi.org/10.1007/s10898-019-00764-3
29. Djelassi, H., Mitsos, A.: A hybrid discretization algorithm with guaranteed feasibility for the global solution of semi-infinite programs. J. Global Optim. **68**(2), 227–253 (2017). https://doi.org/10.1007/s10898-016-0476-7
30. Djelassi, H., Mitsos, A.: Global solution of semi-infinite programs with existence constraints. J. Optim. Theory Appl. **188**(3), 863–881 (2021). https://doi.org/10.1007/s10957-021-01813-2
31. Djelassi, H., Mitsos, A., Stein, O.: Recent advances in nonconvex semi-infinite programming: Applications and algorithms. EURO J. Comput. Optim. **9**(5), 100006 (2021). https://doi.org/10.1016/j.ejco.2021.100006
32. Duarte, B.P.M., Wong, W.K., Atkinson, A.C.: A semi-infinite programming based algorithm for determining T-optimum designs for model discrimination. J. Multivar. Anal. **135**, 11–24 (2015). https://doi.org/10.1016/j.jmva.2014.11.006
33. Fair Isaac Corporation: FICO Xpress-Optimizer, reference manual (2023). https://www.fico.com/fico-xpress-optimization/docs/. Accessed 2024/11/17
34. Falk, J.E., Hoffman, K.: A nonconvex max-min problem. Nav. Res. Logist. Quart. **24**(3), 441–450 (1977). https://doi.org/10.1002/nav.3800240307
35. Fang, S.C., Lin, C.J., Wu, S.Y.: On solving convex quadratic semi-infinite programming problems. Optimization **31**(2), 107–125 (1994). https://doi.org/10.1080/02331939408844009
36. Ferris, M.C., Philpott, A.B.: An interior point algorithm for semi-infinite linear programming. Math. Program. **43**(1–3), 257–276 (1989). https://doi.org/10.1007/BF01582293
37. Fischetti, M., Ljubić, I., Monaci, M., Sinnl, M.: A new general-purpose algorithm for mixed-integer bilevel linear programs. Oper. Res. **65**(6), 1615–1637 (2017). https://doi.org/10.1287/opre.2017.1650
38. Floudas, C.A., Stein, O.: The adaptive convexification algorithm: A feasible point method for semi-infinite programming. SIAM J. Optim. **18**(4), 1187–1208 (2008). https://doi.org/10.1137/060657741
39. GAMS Development Corporation: General algebraic modeling system (GAMS) (2019). http://www.gams.com/
40. Glashoff, K., Gustafson, S.Å.: Linear optimization and approximation: An introduction to the theoretical analysis and numerical treatment of semi-infinite programs, Applied Mathematical Sciences, vol. 45, 1 edn. Springer, New York, New York, NY, USA (1983)
41. Gleixner, A., Hendel, G., Gamrath, G., Achterberg, T., Bastubbe, M., Berthold, T., Christophel, P.M., Jarck, K., Koch, T., Linderoth, J., Lübbecke, M., Mittelmann, H.D., Ozyurt, D., Ralphs, T.K., Salvagnin, D., Shinano, Y.: MIPLIB 2017: Data-driven compilation of the 6th mixed-integer programming library. Math. Program. Comput. (2021). https://doi.org/10.1007/s12532-020-00194-3
42. Goberna, M.A., López, M.A.: Linear semi-infinite optimization. Wiley series in mathematical methods in practice. Wiley, Chichester (1998)
43. Goerigk, M., Kurtz, J., Schmidt, M., Thürauf, J.: Connections between robust and bilevel optimization. Open J. Math. Optim. **6**, 1–17 (2025). https://doi.org/10.5802/ojmo.38
44. Ein Hybridverfahren zur Lösung nichtlinearer semi-infiniter Optimierungsprobleme. Dissertation, Technische Universität Berlin, Berlin (1997)
45. Grossmann, I.E., Sargent, R.W.H.: Optimum design of chemical plants with uncertain parameters. Am. Inst. Chem. Eng. J. **24**(6), 1021–1028 (1978). https://doi.org/10.1002/aic.690240612

46. Guerra-Vázquez, F., Rückmann, J.J., Stein, O., Still, G.: Generalized semi-infinite programming: A tutorial. J. Comput. Appl. Math. **217**(2), 394–419 (2008). https://doi.org/10.1016/j.cam.2007.02.012
47. Gümüş, Z.H.: Reactive distillation column design with vapor/liquid/liquid equilibria. Comput. Chem. Eng. **21**(1–2), 983–988 (1997). https://doi.org/10.1016/S0098-1354(97)00177-4
48. Günzel, H., Jongen, H.T., Stein, O.: On the closure of the feasible set in generalized semi-infinite programming. CEJOR **15**(3), 271–280 (2007). https://doi.org/10.1007/s10100-007-0030-2
49. Guo, F., Sun, X.: Semidefinite programming relaxations for linear semi-infinite polynomial programming. Pacific J. Optim. **16**(3), 395–418 (2020)
50. Guo, F., Zhang, M.: An SDP method for fractional semi-infinite programming problems with SOS-convex polynomials. Optim. Lett. (2023). https://doi.org/10.1007/s11590-023-01974-1
51. Gurobi Optimization, L.: Gurobi Optimizer Reference Manual (2025). https://www.gurobi.com. Accessed 2025/02/21
52. Gustafson, S.Å.: On numerical analysis in semi-infinite programming. In: Balakrishnan, A.V., Thoma, M., Hettich, R. (eds.) Semi-Infinite Programming. Lecture Notes in Control and Information Sciences, vol. 15, pp. 51–65. Springer, Berlin Heidelberg (1979)
53. Gustafson, S.Å., Kortanek, K.O.: Numerical treatment of a class of semi-infinite programming problems. Nav. Res. Logist. Quarterly **20**(3), 477–504 (1973). https://doi.org/10.1002/nav.3800200310
54. Harwood, S.M.: A note on generalized semi-infinite program bounding methods (2019)
55. Harwood, S.M., Barton, P.I.: How to solve a design centering problem. Math. Methods Oper. Res. **86**(1), 215–254 (2017). https://doi.org/10.1007/s00186-017-0591-3
56. Harwood, S.M., Papageorgiou, D.J., Trespalacios, F.: A note on semi-infinite program bounding methods. Optim. Lett. **15**(4), 1485–1490 (2021). https://doi.org/10.1007/s11590-020-01638-4
57. Hettich, R.: Chebyshev approximation by H-polynomials: A numerical method. J. Approx. Theory **17**(1), 97–106 (1976). https://doi.org/10.1016/0021-9045(76)90114-3
58. Hettich, R.: A comparison of some numerical methods for semi-infinite programming. In: Balakrishnan, A.V., Thoma, M., Hettich, R. (eds.) Semi-Infinite Programming. Lecture Notes in Control and Information Sciences, vol. 15, pp. 112–125. Springer, Berlin Heidelberg (1979)
59. Hettich, R.: An implementation of a discretization method for semi-infinite programming. Math. Program. **34**(3), 354–361 (1986). https://doi.org/10.1007/BF01582235
60. Hettich, R., van Honstede, W.: On quadratically convergent methods for semi-infinite programming. In: Balakrishnan, A.V., Thoma, M., Hettich, R. (eds.) Semi-Infinite Programming. Lecture Notes in Control and Information Sciences, vol. 15, pp. 97–111. Springer, Berlin Heidelberg (1979)
61. Hettich, R., Zencke, P.: Numerische Methoden der Approximation und semi-infiniten Optimierung. Teubner Studienbücher: Mathematik. Teubner and Vieweg+Teubner Verlag, Stuttgart (1982)
62. International Business Machines Corporation: IBM ILOG CPLEX v22.1.1 (2022)
63. Jongen, H.T., Rückmann, J.J., Stein, O.: Generalized semi-infinite optimization: A first order optimality condition and examples. Math. Program. **83**(1–3), 145–158 (1998). https://doi.org/10.1007/BF02680555
64. Jungen, D.: Deterministic global and hierarchical optimization for experimental design. Dissertation, RWTH Aachen University, Aachen, Germany (2025). https://doi.org/10.18154/RWTH-2025-10903
65. Jungen, D., Djelassi, H., Mitsos, A.: Adaptive discretization-based algorithms for semi-infinite programs with unbounded variables. Math. Methods Oper. Res. **96**(1), 83–112 (2022). https://doi.org/10.1007/s00186-022-00792-y
66. Jungen, D., Zingler, A., Djelassi, H., Mitsos, A.: Global optimization of bilevel programs and (generalized) semi-infinite programs – an open-source platform. In: SIAM Conference on Optimization (OP23). Seattle, WA, USA (2023). Conference presentation
67. Jungen, D., Zingler, A., Djelassi, H., Mitsos, A.: libDIPS – Discretization-based semi-Infinite and bilevel Programming Solvers. Optimization Online (2023). https://optimization-online.org/?p=24914
68. Jungen, D., Zingler, A., Djelassi, H., Mitsos, A.: Open-source software for the global optimization of bilevel programs and (generalized) semi-infinite programs. In: International Conference on Operations Research 2023. Hamburg, Germany (2023). Conference presentation
69. Kanzi, N.: Lagrange multiplier rules for non-differentiable DC generalized semi-infinite programming problems. J. Global Optim. **56**(2), 417–430 (2013). https://doi.org/10.1007/s10898-011-9828-5
70. Kleniati, P.M., Adjiman, C.S.: Branch-and-sandwich: a deterministic global optimization algorithm for optimistic bilevel programming problems. Part II: Convergence analysis and numerical results. J. Global Optim. **60**(3), 459–481 (2014). https://doi.org/10.1007/s10898-013-0120-8

71. Kleniati, P.M., Adjiman, C.S.: A generalization of the branch-and-sandwich algorithm: From continuous to mixed-integer nonlinear bilevel problems. Comput. Chem. Eng. **72**, 373–386 (2015). https://doi.org/10.1016/j.compchemeng.2014.06.004
72. Kostyukova, O.I., Tchemisova, T.V.: Sufficient optimality conditions for convex semi-infinite programming. Optim. Method. Softw. **25**(2), 279–297 (2010). https://doi.org/10.1080/10556780902992803
73. Küfer, K.H., Stein, O., Winterfeld, A.: Semi-infinite optimization meets industry: A deterministic approach to gemstone cutting. SIAM News 41 (2008)
74. Lemonidis, P.: Global optimization algorithms for semi-infinite and generalized semi-infinite programs. Ph.D. thesis, Massachusetts Institute of Technology, USA (2008)
75. Leon, T., Vercher, E.: A purification algorithm for semi-infinite programming. Eur. J. Oper. Res. **57**(3), 412–420 (1992). https://doi.org/10.1016/0377-2217(92)90353-B
76. Li, D.H., Qi, L., Tam, J., Wu, S.Y.: A smoothing Newton method for semi-infinite programming. J. Global Optim. **30**(2–3), 169–194 (2004). https://doi.org/10.1007/s10898-004-8266-z
77. Li, S.J.: Semi-infinite programming and semi-definite optimization problems. Ph.D. thesis, Hong Kong Polytechnic University (2003)
78. Lin, C.J., Fang, S.C., Wu, S.Y.: A dual affine scaling based algorithm for solving linear semi-infinite programming problems. In: Pardalos, P., Horst, R., Du, D.Z., Sun, J. (eds.) Advances in Optimization and Approximation, Nonconvex Optimization and Its Applications, vol. 1, pp. 217–234. Springer, US, Boston, MA, USA (1994)
79. Liu, W., Wang, C.: A smoothing Levenberg-Marquardt method for generalized semi-infinite programming. Comput. Appl. Math. **32**(1), 89–105 (2013). https://doi.org/10.1007/s40314-013-0013-y
80. Liu, Y., Teo, K.L., Ito, S.: A dual parameterization approach to linear-quadratic semi-infinite programming problems. Optim. Method. Softw. **10**(3), 471–495 (1999). https://doi.org/10.1080/10556789908805725
81. Lo Bianco, C.G., Piazzi, A.: A hybrid algorithm for infinitely constrained optimization. Int. J. Syst. Sci. **32**(1), 91–102 (2001). https://doi.org/10.1080/00207720121051
82. López, M., Still, G.: Semi-infinite programming. Eur. J. Oper. Res. **180**(2), 491–518 (2007). https://doi.org/10.1016/j.ejor.2006.08.045
83. Marendet, A., Goldsztejn, A., Chabert, G., Jermann, C.: A standard branch-and-bound approach for nonlinear semi-infinite problems. Eur. J. Oper. Res. **282**(2), 438–452 (2020). https://doi.org/10.1016/j.ejor.2019.10.025
84. Mehrotra, S., Papp, D.: A cutting surface algorithm for semi-infinite convex programming with an application to moment robust optimization. SIAM Journal on Optimization: A Publication of the Society for Industrial and Applied Mathematics **24**(4), 1670–1697 (2014). https://doi.org/10.1137/130925013
85. Misener, R., Floudas, C.A.: ANTIGONE: Algorithms for coNTinuous / Integer Global Optimization of Nonlinear Equations. J. Global Optim. **59**(2), 503–526 (2014). https://doi.org/10.1007/s10898-014-0166-2
86. Mitsos, A.: Man-portable power generation devices: Product design and supporting algorithms. Ph.D. thesis, Massachusetts Institute of Technology (2006)
87. Mitsos, A.: Test set of semi-infinite programs (2009). https://www.avt.rwth-aachen.de/cms/AVT/Forschung/Systemverfahrenstechnik/~kpdo/A-Test-Set-of-Semi-Infinite-Programs/?lidx=1. Accessed 2023/08/17
88. Mitsos, A.: Global solution of nonlinear mixed-integer bilevel programs. J. Global Optim. **47**(4), 557–582 (2010). https://doi.org/10.1007/s10898-009-9479-y
89. Mitsos, A.: Global optimization of semi-infinite programs via restriction of the right-hand side. Optimization **60**(10–11), 1291–1308 (2011). https://doi.org/10.1080/02331934.2010.527970
90. Mitsos, A., Lemonidis, P., Barton, P.I.: Global solution of bilevel programs with a nonconvex inner program. J. Global Optim. **42**(4), 475–513 (2008). https://doi.org/10.1007/s10898-007-9260-z
91. Mitsos, A., Lemonidis, P., Lee, C.K., Barton, P.I.: Relaxation-based bounds for semi-infinite programs. SIAM Journal on Optimization: A Publication of the Society for Industrial and Applied Mathematics **19**(1), 77–113 (2008). https://doi.org/10.1137/060674685
92. Mitsos, A., Tsoukalas, A.: Global optimization of generalized semi-infinite programs via restriction of the right hand side. J. Global Optim. **61**(1), 1–17 (2015). https://doi.org/10.1007/s10898-014-0146-6

93. Najman, J., Bongartz, D., Mitsos, A.: Relaxations of thermodynamic property and costing models in process engineering. Comput. Chem. Eng. **130**, 106571 (2019). https://doi.org/10.1016/j.compchemeng.2019.106571
94. Pang, L.P., Lv, J., Wang, J.H.: Constrained incremental bundle method with partial inexact oracle for nonsmooth convex semi-infinite programming problems. Comput. Optim. Appl. **64**(2), 433–465 (2016). https://doi.org/10.1007/s10589-015-9810-0
95. Pang, L.P., Wu, Q.: A feasible proximal bundle algorithm with convexification for nonsmooth, nonconvex semi-infinite programming. Numerical Algorithms **90**(1), 387–422 (2022). https://doi.org/10.1007/s11075-021-01192-9
96. Paulavičius, R., Adjiman, C.S.: BASBLib – a library of bilevel test problems. Comput. Chem. Eng. **132**, 106609 (2020). https://doi.org/10.5281/zenodo.3266835
97. Polak, E., Qi, L., Sun, D.: First-order algorithms for generalized semi-infinite min-max problems. Comput. Optim. Appl. **13**(1/3), 137–161 (1999). https://doi.org/10.1023/A:1008660924636
98. Powell, M.J.D.: Karmarkar's algorithm: A view from nonlinear programming (1989). https://ir.canterbury.ac.nz/items/50cd806f-63c4-456b-a66e-132f78653b64. Accessed 2023/08/17
99. Price, C.J.: Non-linear semi-infinite programming. Ph.D. thesis, University of Canterbury, New Zealand (1992). https://doi.org/10.26021/8870
100. Price, C.J., Coope, I.D.: Numerical experiments in semi-infinite programming. Comput. Optim. Appl. **6**(2), 169–189 (1996). https://doi.org/10.1007/BF00249645
101. Qi, L., Wu, S.Y., Zhou, G.: Semismooth Newton methods for solving semi-infinite programming problems. J. Global Optim. **27**(2/3), 215–232 (2003). https://doi.org/10.1023/A:1024814401713
102. Reemtsen, R., Rückmann, J.J. (eds.): Semi-Infinite Programming, Nonconvex Optimization and Its Applications, vol. 25. Springer, US, Boston, MA, USA (1998). https://doi.org/10.1007/978-1-4757-2868-2
103. Remez, E.I.: General computational methods of Chebyshev approximation: The problems with linear real parameters. Translation series, AEC-tr-4491. U.S. Atomic Energy Comission. Division of Technical Information, Oak Ridge, Tennessee, USA (1962)
104. Royset, J.O., Polak, E., Kiureghian, A.: Adaptive approximations and exact penalization for the solution of generalized semi-infinite min-max problems. SIAM Journal on Optimization: A Publication of the Society for Industrial and Applied Mathematics **14**(1), 1–34 (2003). https://doi.org/10.1137/S1052623402406777
105. Rückmann, J.J., Shapiro, A.: First-order optimality conditions in generalized semi-infinite programming. J. Optim. Theory Appl. **101**(3), 677–691 (1999). https://doi.org/10.1023/A:1021746305759
106. Rückmann, J.J., Shapiro, A.: Second-order optimality conditions in generalized semi-infinite programming. Set-Valued Var. Anal. **9**(1/2), 169–186 (2001). https://doi.org/10.1023/A:1011239607220
107. Rückmann, J.J., Shapiro, A.: Augmented Lagrangians in semi-infinite programming. Math. Program. **116**(1–2), 499–512 (2009). https://doi.org/10.1007/s10107-007-0115-7
108. Rudnick-Cohen, E., Herrmann, J.W., Azarm, S.: Non-convex feasibility robust optimization via scenario generation and local refinement. J. Mech. Des. **142**(5), 051703 (2020). https://doi.org/10.1115/1.4044918
109. Sahinidis, N.V.: BARON 2023.6.23: Global Optimization of Mixed-Integer Nonlinear Programs, User's Manual (2023). http://www.minlp.com/downloads/docs/baron%20manual.pdf. Accessed 2023/09/04
110. Schwientek, J., Seidel, T., Küfer, K.H.: A transformation-based discretization method for solving general semi-infinite optimization problems. Math. Methods Oper. Res. **93**(1), 83–114 (2021). https://doi.org/10.1007/s00186-020-00724-8
111. Seidel, T., Küfer, K.H.: An adaptive discretization method solving semi-infinite optimization problems with quadratic rate of convergence. Optimization **71**(8), 2211–2239 (2022). https://doi.org/10.1080/02331934.2020.1804566
112. Seidel, T., Schwientek, J.: GSIPLib&Gen: A library and generator of general semi-infinite programming test problems: Version 1.0 (2017). https://www.itwm.fraunhofer.de/en/departments/optimization/products-and-services/gsip-lib-and-gen.html. Accessed 2023/04/19
113. Selassie, A.G.W.: A coarse solution of generalized semi-infinite optimization problems via robust analysis of marginal functions and global optimization. Dissertation, Technischen Universitat Ilmenau (2004)

114. Sinha, A., Malo, P., Deb, K.: A review on bilevel optimization: From classical to evolutionary approaches and applications. IEEE Trans. Evol. Comput. **22**(2), 276–295 (2018). https://doi.org/10.1109/TEVC.2017.2712906
115. Spjøtvold, J., Tøndel, P., Johansen, T.A.: A method for obtaing continuos solutions to multiparametric linear programs. IFAC proceedings volumes **38**(1), 253–258 (2005). https://doi.org/10.3182/20050703-6-CZ-1902.00903
116. Stein, O.: Bi-Level Strategies in Semi-Infinite Programming, Nonconvex Optimization and Its Applications, vol. 71. Springer, US, Boston, MA (2003)
117. Stein, O., Steuermann, P.: The adaptive convexification algorithm for semi-infinite programming with arbitrary index sets. Math. Program. **136**(1), 183–207 (2012). https://doi.org/10.1007/s10107-012-0556-5
118. Stein, O., Still, G.: On generalized semi-infinite optimization and bilevel optimization. Eur. J. Oper. Res. **142**(3), 444–462 (2002). https://doi.org/10.1016/S0377-2217(01)00307-1
119. Still, G.: Generalized semi-infinite programming: Theory and methods. Eur. J. Oper. Res. **119**(2), 301–313 (1999). https://doi.org/10.1016/S0377-2217(99)00132-0
120. Still, G.: Discretization in semi-infinite programming: The rate of convergence. Math. Program. **91**(1), 53–69 (2001). https://doi.org/10.1007/s101070100239
121. Still, G.: Generalized semi-infinite programming: Numerical aspects. Optimization **49**(3), 223–242 (2001). https://doi.org/10.1080/02331930108844531
122. Still, G.: Optimization problems with infinitely many constraints. Buletinul ştiinţific al Universitatii Baia Mare, Seria B, Fascicola matematică-informatică **18**(2), 343–354 (2002)
123. Streit, R.L., Nuttall, A.H.: A general Chebyshev complex function approximation procedure and an application to beamforming. J. Acoust. Soc. Am. **72**(1), 181–190 (1982). https://doi.org/10.1121/1.388002
124. Su, K., Xu, C., Ren, L.: Filter trust region method for nonlinear semi-infinite programming problem. Math. Probl. Eng. **2018**, 1–9 (2018). https://doi.org/10.1155/2018/3921592
125. Swaney, R.E., Grossmann, I.E.: An index for operational flexibility in chemical process design. Part I: Formulation and theory. AIChE J. **31**(4), 621–630 (1985). https://doi.org/10.1002/aic.690310412
126. Tahernejad, S., Ralphs, T.K., DeNegre, S.T.: A branch-and-cut algorithm for mixed integer bilevel linear optimization problems and its implementation. Math. Program. Comput. **12**(4), 529–568 (2020). https://doi.org/10.1007/s12532-020-00183-6
127. Tanaka, Y., Fukushima, M., Ibaraki, T.: A globally convergent SQP method for semi-infinite nonlinear optimization. J. Comput. Appl. Math. **23**(2), 141–153 (1988). https://doi.org/10.1016/0377-0427(88)90276-2
128. Teo, K.L., Yang, X.Q., Jennings, L.S.: Computational discretization algorithms for functional inequality constrained optimization. Ann. Oper. Res. **98**(1/4), 215–234 (2000). https://doi.org/10.1023/A:1019260508329
129. Tezel Özturan, A.: Solving generalized semi-infinite programming problems with a trust region method. Acta Phys. Pol., A **128**(2B), B-93-B−97 (2015). https://doi.org/10.12693/APhysPolA.128.B-93
130. Tichatschke, R., Nebeling, V.: A cutting-plane method for quadratic semi infinite programming problems. Optimization **19**(6), 803–817 (1988). https://doi.org/10.1080/02331938808843393
131. Tsoukalas, A., Rustem, B.: A feasible point adaptation of the Blankenship and Falk algorithm for semi-infinite programming. Optim. Lett. **5**(4), 705–716 (2011). https://doi.org/10.1007/s11590-010-0236-4
132. Tsoukalas, A., Rustem, B., Pistikopoulos, E.N.: A global optimization algorithm for generalized semi-infinite, continuous minimax with coupled constraints and bi-level problems. J. Global Optim. **44**(2), 235–250 (2009). https://doi.org/10.1007/s10898-008-9321-y
133. Vaz, A.I.F., Fernandes, E., Gomes, M.: NSIPS: Nonlinear Semi-Infinite Programming Solver (2004)
134. Vaz, A.I.F., Fernandes, E.M.G.P., Gomes, M.P.S.F.: SIPAMPL. ACM Trans. Math. Softw. **30**(1), 47–61 (2004). https://doi.org/10.1145/974781.974784
135. Vázquez, F.G., Rückmann, J.J.: Extensions of the Kuhn-Tucker constraint qualification to generalized semi-infinite programming. SIAM Journal on Optimization: A Publication of the Society for Industrial and Applied Mathematics **15**(3), 926–937 (2005). https://doi.org/10.1137/S1052623403431500
136. Wächter, A., Biegler, L.T.: On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. Math. Program. **106**(1), 25–57 (2006). https://doi.org/10.1007/s10107-004-0559-y

137. Watson, G.A.: A multiple exchange algorithm for multivariate Chebyshev approximation. SIAM J. Numer. Anal. **12**(1), 46–52 (1975). https://doi.org/10.1137/0712004
138. Watson, G.A.: Numerical experiments with globally convergent methods for semi-infinite programming problems. In: Fiacco, A.V., Kortanek, K.O. (eds.) Semi-Infinite Programming and Applications. Lecture Notes in Economics and Mathematical Systems, pp. 193–205. Springer, Berlin Heidelberg, Berlin, Heidelberg (1983)
139. Wiesemann, W., Tsoukalas, A., Kleniati, P.M., Rustem, B.: Pessimistic bilevel optimization. SIAM J. Optim. **23**(1), 353–380 (2013). https://doi.org/10.1137/120864015
140. Wilhelm, M.E., Stuber, M.D.: EAGO.jl: easy advanced global optimization in Julia. Optim. Methods Softw. **37**(2), 425–450 (2022). https://doi.org/10.1080/10556788.2020.1786566
141. Winterfeld, A.: Application of general semi-infinite programming to lapidary cutting problems. Eur. J. Oper. Res. **191**(3), 838–854 (2008). https://doi.org/10.1016/j.ejor.2007.01.057
142. Wu, S.Y., Li, D.H., Qi, L., Zhou, G.: An iterative method for solving KKT system of the semi-infinite programming. Optim. Method. Softw. **20**(6), 629–643 (2005). https://doi.org/10.1080/10556780500094739
143. Xu, Y., Sun, W., Qi, L.: On solving a class of linear semi-infinite programming by SDP method. Optimization pp. 1–14 (2013). https://doi.org/10.1080/02331934.2013.793325
144. Žaković, S., Rustem, B.: Semi-infinite programming and applications to minimax problems. Ann. Oper. Res. **124**(1–4), 81–110 (2003). https://doi.org/10.1023/B:ANOR.0000004764.76984.30
145. Zhang, L., Wu, S.Y., López, M.A.: A new exchange method for convex semi-infinite programming. SIAM Journal on Optimization: A Publication of the Society for Industrial and Applied Mathematics **20**(6), 2959–2977 (2010). https://doi.org/10.1137/090767133
146. Zhou, J.L., Tits, A.L.: An SQP algorithm for finely discretized continuous minimax problems and other minimax problems with many objective functions. SIAM Journal on Optimization: A Publication of the Society for Industrial and Applied Mathematics **6**(2), 461–487 (1996). https://doi.org/10.1137/0806025
147. Zhou, S., Zemkoho, A.B.: BiOpt: Bilevel optimization toolboxes (2021). https://biopt.github.io/. Accessed 2024/11/29
148. Zhou, S., Zemkoho, A.B., Tin, A.: BOLIB: Bilevel Optimization LIBrary of Test Problems. In: Dempe, S., Zemkoho, A.B. (eds.) Bilevel Optimization, Springer Optimization and Its Applications, vol. 161, pp. 563–580. Springer, Cham, Switzerland (2020)
149. Zingler, A.: Adaptive discretization methods for the global solution of semi-infinite optimization problems. Dissertation, RWTH Aachen University, Aachen, Germany (2025). https://doi.org/10.18154/RWTH-2025-05006
150. Zingler, A., Fliscounakis, S., Panciatici, P., Mitsos, A.: Optimizing flexibility in power systems by maximizing the region of manageable uncertainties. Optimization and Engineering pp. 1–23 (2025). https://doi.org/10.1007/s11081-025-09958-z
151. Zingler, A., Jungen, D., Djelassi, H., Mitsos, A.: libALE – a library for algebraic logical expression trees (2022). https://git.rwth-aachen.de/avt.svt/public/libale