libDIPS – Discretization-Based Semi-Infinite and Bilevel Programming Solvers

Daniel Jungen[†,◇], Aron Zingler[†,◇], Hatim Djelassi[†], and Alexander Mitsos[‡,†,§,*]
date: December 4, 2023
Abstract: We consider several hierarchical optimization programs: (generalized) semi-infinite and existence-constrained semi-infinite programs, minmax, and bilevel programs. Multiple adaptive discretization-based algorithms have been published for these program classes in recent decades. However, rigorous numerical performance comparisons between these algorithms are lacking. Indeed, if numerical comparisons are provided at all, they usually compare a small selection of algorithms on small benchmark test sets, on different platforms, and with different subsolvers, which are needed during the solution. Additionally, some algorithms have hyperparameters, which impedes 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 main benefit of libDIPS is that it lets the user flexibly change between the implemented solvers within one program class and switch between the available subsolvers. ii) We compile an extensive benchmark test set for (generalized) semi-infinite, minmax, and bilevel programs, 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 on our benchmark test set and identify tradeoffs in the hyperparameters tuning.

Keywords— semi-infinite programming, generalized semi-infinite programming, bilevel programming, adaptive discretization, benchmark, software

1 Introduction

Generalized semi-infinite and semi-infinite programs ((G)SIPs), existence-constrained semi-infinite programs (ESIPs), minmax programs (MINMAX), and bilevel programs (BLPs) are hierarchical optimization programs that occur in many applications, e.g., robust optimization [9], flexibility analysis [112], gemstone cutting [127], and, thermodynamics [20, 42].

A reliable and fast solution to such optimization programs is paramount but also very challenging: the lower-level optimization problem must be solved globally even to check the feasibility of a given candidate solution point. If the lower-level problems of (G)SIPs, MINMAX, or BLPs are convex and satisfy some regularity conditions, the hierarchical program can be reformulated as a single-level problem. However, in many applications, the lower-level problems are nonconvex.

Multiple theoretical approaches for the global solution of hierarchical programs without convexity assumptions have been developed over the past decades, with substantial recent advances. These include classical discretization methods, adaptive discretization-based approaches and adaptions thereof, overestimation methods using interval methods and relaxation methods; optimal value function approaches; and relaxation-based branch-and-bound methods [12–14, 26, 32, 35, 71, 73, 77, 78, 78, 98, 104, 117]. See [29] for more details.

Here, we focus on the adaptive discretization approaches. A primary benefit of these adaptive discretization approaches is that they can utilize well-established optimization solvers mainly as a black box. They have also been successfully applied in

[†] Process Systems Engineering (AVT.SVT), RWTH Aachen University, 52074 Aachen, Germany
[‡] JARA-CSD, 52056 Aachen, Germany
[§] Institute of Energy and Climate Research: Energy Systems Engineering (IEK-10), Forschungszentrum Jülich GmbH, 52425 Jülich, Germany.
* Corresponding author: A. Mitsos
E-mail: amitsos@alum.mit.edu
[◇] D.J. and A.Z. contributed equally to this work.

research and industry [63]. Over the last decades, numerous adaptive discretization-based algorithms have been published to solve the different program classes, e.g., refer to Table 2.

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

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, [18, 60, 73] compared the numerical performance of their approaches against the algorithms proposed by [26, 77, 79]. For their comparisons, [18] used their own implementations of [77], [73] used the GAMS implementation provided by [26], and [60] used the data reported by [79].

The additional effort to reimplement the respective adaptive discretization algorithm, not having access to the required commercially distributed subsolvers, or the need for a comparison that is not possible based solely on the original publication might have discouraged other researchers from using them as a point of comparison. An alternative to implementation would be to use one of the publicly available solvers for hierarchical programs, listed in Table 1 with their respective capabilities. A potential reason preventing these from being used in a comparison are limitations placed on the problem type in the upper- or lower-level, the lack of a deterministic global solution, or that the relevant problem class, i.e., GSIP, SIP, or BLP, is not supported. Additionally, some solvers are commercially distributed or use 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, MINMAX, and BLPs with upper- and lower-levels of MINLP type. Our software is not only freely available and easy to use but rather is also a suitable framework to implement existing and new algorithms, whereby discretization-based algorithms will be the easiest to implement. Additionally, we compile an extensive test set of problem instances to compare the algorithms. The combination of libDIPS and the test set arguably makes further comparisons between algorithms easier for other researchers and makes the solution of hierarchical programs more accessible.

In Section 2, for completeness, we shortly review the program classes (G)SIP, ESIP, MINMAX, and BLP. In Section 3, we recapitulate the main ideas of adaptive discretization-based algorithms for solving these program classes. We then introduce in Section 4 an open-source C++ library, called libDIPS, which contains implementations of the algorithms reviewed in Section 3. In Section 5, we compile a benchmark test set consisting of unified existing benchmark test sets and other test problems found in the literature. Section 6 covers performance tests. We utilize the benchmark test set to compare the overall performance of the implemented solvers for 'best case' hyperparameters for the benchmark test set. Additionally, we investigate the sensitivity of the different solvers to their hyperparameters. We summarize our work in Section 7 and propose future work.

	SIP	GSIP	BLP	Det. Global	Type (upper-lower)*	Algorithm Type
EAGO [126]	✓			✓	MINLP-MINLP	Adaptive discretization
NSIPS [119, 120]	✓				NLP-unconstrained NLP	Discretization, transcription to integral
SIP with QP-LL [18]	✓			✓	convex NLP-QP	Adaptive discretization and Dualization
IbexSIP [73]	✓			✓	NLP-NLP	Interval Method with Branch & Bound
BASBL [61, 84]			✓	✓	MINLP-MINLP	Branch & Sandwich
B-POP [5]			✓	✓	MIQP-MIQP	Multi-parametric
GAMS EMP [36]			✓	✓	NLP-convex NLP	KKT-based MPEC
libDIPS (this work)	✓	✓	✓	✓	MINLP-MINLP	Adaptive discretization

Table 1: Overview of available solvers for hierarchical programs. The solvers by [4, 11] are published but not publicly available. *: Type describes the problem type of the upper- and lower-level problem, respectively.

2 The Program Classes SIP, GSIP, ESIP, MINMAX, and BLP

In the following, we state the problem formulations and notations used for the program classes (G)SIP, ESIP, MINMAX, and BLP. While we highlight important aspects of each program class, we discuss only the problem classes and their formulations that are directly tractable with libDIPS. As an example, the algorithm for *pessimistic* bilevel problems from [125] can easily be implemented in the discussed software framework, but we do not discuss this problem class, because the algorithm is currently not implemented. Furthermore, we focus on features related to the algorithmic ideas introduced in Section 3. Most program classes are well-established, and we thus refer to surveys for a more thorough discussion [29, 41, 83, 101]. For the problem class of ESIPs we refer to [27].

Throughout the manuscript, we write vector-valued symbols in bold font and sets in calligraphic font. While non-compact host sets can theoretically be handled with adaptive discretization algorithms under certain assumptions [58], we assume compact host sets herein and in our open-source software libDIPS. We assume that all occurring functions are continuous on their respective host sets.

SIPs are formulated as

$$\begin{aligned} \min_{\boldsymbol{x}\in\mathcal{X}} \quad & f(\boldsymbol{x}) \\ \text{s.t.} \quad & \boldsymbol{g}^u(\boldsymbol{x},\boldsymbol{y}) \leqslant \boldsymbol{0},\ \forall\ \boldsymbol{y}\in\mathcal{Y} \\ & \mathcal{X} := \left\{\boldsymbol{x}\in\left[\boldsymbol{x}^{lb},\boldsymbol{x}^{ub}\right]\subsetneq\mathbb{R}^{n_x} : \boldsymbol{v}^{iu}(\boldsymbol{x})\leqslant\boldsymbol{0},\ \boldsymbol{v}^{eu}(\boldsymbol{x})=\boldsymbol{0}\right\} \\ & \mathcal{Y} := \left\{\boldsymbol{y}\in\left[\boldsymbol{y}^{lb},\boldsymbol{y}^{ub}\right]\subsetneq\mathbb{R}^{n_y} : \boldsymbol{v}^{il}(\boldsymbol{y})\leqslant\boldsymbol{0},\ \boldsymbol{v}^{el}(\boldsymbol{y})=\boldsymbol{0}\right\}; \end{aligned} \tag{SIP}$$

with the objective function $f:\mathbb{R}^{n_x}\to\mathbb{R}$, the semi-infinite constraint function $\boldsymbol{g}^u:\mathbb{R}^{n_x}\times\mathbb{R}^{n_y}\to\mathbb{R}^{n_{gu}}$, non-coupling upper-level equality and inequality constraints $\boldsymbol{v}^{iu}:\mathbb{R}^{n_x}\to\mathbb{R}^{n_{viu}}$ and $\boldsymbol{v}^{eu}:\mathbb{R}^{n_x}\to\mathbb{R}^{n_{veu}}$, non-coupling lower-level equality and inequality constraints $\boldsymbol{v}^{il}:\mathbb{R}^{n_y}\to\mathbb{R}^{n_{vil}}$ and $\boldsymbol{v}^{el}:\mathbb{R}^{n_y}\to\mathbb{R}^{n_{vel}}$. We call $\boldsymbol{x}$ the upper-level variables and $\boldsymbol{y}$ the lower-level variables. The set of all considered values of the lower-level variables, the so-called lower-level feasible set $\mathcal{Y}$, is a set of infinity cardinality.

We consider MINMAX of the form

$$\begin{aligned} \min_{\boldsymbol{x}\in\mathcal{X}} \quad & \max_{y\in\mathcal{Y}} f(\boldsymbol{x},\boldsymbol{y}) \\ \text{s.t.} \quad & \mathcal{X} := \left\{\boldsymbol{x}\in\left[\boldsymbol{x}^{lb},\boldsymbol{x}^{ub}\right]\subsetneq\mathbb{R}^{n_x} : \boldsymbol{v}^{iu}(\boldsymbol{x})\leqslant\boldsymbol{0},\ \boldsymbol{v}^{eu}(\boldsymbol{x})=\boldsymbol{0}\right\} \\ & \mathcal{Y} := \left\{\boldsymbol{y}\in\left[\boldsymbol{y}^{lb},\boldsymbol{y}^{ub}\right]\subsetneq\mathbb{R}^{n_y} : \boldsymbol{v}^{il}(\boldsymbol{y})\leqslant\boldsymbol{0},\ \boldsymbol{v}^{el}(\boldsymbol{y})=\boldsymbol{0}\right\}; \end{aligned} \tag{MINMAX}$$

with the objective function $f:\mathbb{R}^{n_x}\times\mathbb{R}^{n_y}\to\mathbb{R}$ and all other functions as defined in (SIP). (MINMAX) is a specialization of (SIP), as can be seen from the reformulation

$$\begin{aligned} \min_{\boldsymbol{x}\in\mathcal{X},t\in\mathbb{R}} \quad & t \\ \text{s.t.} \quad & f(\boldsymbol{x},\boldsymbol{y}) - t \leqslant 0\ \forall\ \boldsymbol{y}\in\mathcal{Y}\,. \end{aligned} \tag{MINMAX-REF}$$

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

$$\begin{aligned} \min_{\boldsymbol{x}\in\mathcal{X}} \quad & f(\boldsymbol{x}) \\ \text{s.t.} \quad & \boldsymbol{g}^u(\boldsymbol{x},\boldsymbol{y}) \leqslant \boldsymbol{0},\ \forall\ \boldsymbol{y}\in\mathcal{Y}(\boldsymbol{x}) \\ & \mathcal{X} := \left\{\boldsymbol{x}\in\left[\boldsymbol{x}^{lb},\boldsymbol{x}^{ub}\right]\subsetneq\mathbb{R}^{n_x} : \boldsymbol{v}^{iu}(\boldsymbol{x})\leqslant\boldsymbol{0},\ \boldsymbol{v}^{eu}(\boldsymbol{x})=\boldsymbol{0}\right\} \\ & \mathcal{Y}(\boldsymbol{x}) := \left\{\boldsymbol{y}\in\left[\boldsymbol{y}^{lb},\boldsymbol{y}^{ub}\right]\subsetneq\mathbb{R}^{n_y} : \boldsymbol{g}^l(\boldsymbol{x},\boldsymbol{y})\leqslant\boldsymbol{0},\ \boldsymbol{v}^{il}(\boldsymbol{y})\leqslant\boldsymbol{0},\ \boldsymbol{v}^{el}(\boldsymbol{y})=\boldsymbol{0}\right\}, \end{aligned} \tag{GSIP}$$

with the coupling lower-level inequality constraint function $\boldsymbol{g}^l:\mathbb{R}^{n_x}\times\mathbb{R}^{n_y}\to\mathbb{R}^{n_{gl}}$ and all other functions as defined in (SIP).

ESIPs are another generalization of SIPs. ESIPs allow the modeling of an existence constraint instead of a semi-infinite constraint:

$$\begin{aligned}\min_{\boldsymbol{x}\in\mathcal{X}} \quad & f(\boldsymbol{x}) \\ \text{s.t.} \quad & \forall\, \boldsymbol{y} \in \mathcal{Y}[\exists \boldsymbol{z} \in \mathcal{Z} : \boldsymbol{g}^u(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}) \leqslant \boldsymbol{0}] \\ & \mathcal{X} := \left\{\boldsymbol{x} \in \left[\boldsymbol{x}^{lb}, \boldsymbol{x}^{ub}\right] \subsetneq \mathbb{R}^{n_x} : \boldsymbol{v}^{iu}(\boldsymbol{x}) \leqslant \boldsymbol{0},\ \boldsymbol{v}^{eu}(\boldsymbol{x}) = \boldsymbol{0}\right\} \\ & \mathcal{Y} := \left\{\boldsymbol{y} \in \left[\boldsymbol{y}^{lb}, \boldsymbol{y}^{ub}\right] \subsetneq \mathbb{R}^{n_y} : \boldsymbol{v}^{il}(\boldsymbol{y}) \leqslant \boldsymbol{0},\ \boldsymbol{v}^{el}(\boldsymbol{y}) = \boldsymbol{0}\right\} \\ & \mathcal{Z} := \left\{\boldsymbol{z} \in \left[\boldsymbol{z}^{lb}, \boldsymbol{z}^{ub}\right] \subsetneq \mathbb{R}^{n_z} : \boldsymbol{v}^{ie}(\boldsymbol{z}) \leqslant \boldsymbol{0},\ \boldsymbol{v}^{ee}(\boldsymbol{z}) = \boldsymbol{0}\right\};\end{aligned} \tag{ESIP}$$

with $\boldsymbol{g}^u : \mathbb{R}^{n_x} \times \mathbb{R}^{n_y} \times \mathbb{R}^{n_z} \to \mathbb{R}^{n_{gu}}$, $\boldsymbol{v}^{ie} : \mathbb{R}^{n_z} \to \mathbb{R}^{n_{vie}}$, $\boldsymbol{v}^{ee} : \mathbb{R}^{n_z} \to \mathbb{R}^{n_{vee}}$, and all other functions as defined in (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. As discussed later, ESIPs can be considered as SIPs with another hierarchical problem embedded. In this sense, they can be considered a three-level optimization problem. Generalizations to cases where the lower-level feasible set $\mathcal{Y}$ depends on the upper-level variables $\boldsymbol{x}$, or the feasible set of existence-constrained variables $\mathcal{Z}$ depends on the lower-level variables $\boldsymbol{y}$ or the upper-level variables $\boldsymbol{x}$ are straightforward, but not yet implemented in libDIPS. See [27] for these extensions.

The optimistic formulation of a BLP reads

$$\begin{aligned}\min_{\boldsymbol{x}\in\mathcal{X},\boldsymbol{y}\in\mathcal{Y}} \quad & f(\boldsymbol{x},\boldsymbol{y}) \\ \text{s.t.} \quad & \boldsymbol{g}^u(\boldsymbol{x},\boldsymbol{y}) \leqslant \boldsymbol{0} \\ & \boldsymbol{y} \in \arg\min_{\boldsymbol{z}\in\mathcal{Y}(\boldsymbol{x})} \ h(\boldsymbol{x},\boldsymbol{z}) \\ & \mathcal{Y}(\boldsymbol{x}) := \left\{\boldsymbol{y} \in \left[\boldsymbol{y}^{lb}, \boldsymbol{y}^{ub}\right] \subsetneq \mathbb{R}^{n_y} : \boldsymbol{g}^l(\boldsymbol{x},\boldsymbol{y}) \leqslant \boldsymbol{0},\ \boldsymbol{v}^{il}(\boldsymbol{y}) \leqslant \boldsymbol{0},\ \boldsymbol{v}^{el}(\boldsymbol{y}) = \boldsymbol{0}\right\}\ ;\end{aligned} \tag{BLP}$$

with the upper-level objective function $f : \mathbb{R}^{n_x} \times \mathbb{R}^{n_y} \to \mathbb{R}$, the lower-level objective function $h : \mathbb{R}^{n_x} \times \mathbb{R}^{n_y} \to \mathbb{R}$, and all other functions defined as in (SIP). (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 the global solution of an embedded optimization problem 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 MINMAX problems 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 only addressing the most general problem class. BLPs and GSIPs are closely related: if $\nexists \bar{\boldsymbol{x}} \in \mathcal{X}$ such that $\mathcal{Y}(\bar{\boldsymbol{x}}) = \varnothing$, (BLP) and (GSIP) are equivalent [105]. However, there is a distinctive difference if a point $\bar{\boldsymbol{x}}$ exists which leads to an empty lower-level feasible set $\mathcal{Y}(\bar{\boldsymbol{x}}) = \varnothing$; the point $\bar{\boldsymbol{x}}$ is infeasible in (BLP), while it is feasible in (GSIP). The close relation of the problem classes is also mirrored in a close connection in 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 programs covered in Section 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 one software.

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), (MINMAX), and (BLP) compared to standard nonlinear optimization problems is the infinite cardinality of the set $\mathcal{Y}$ and $\mathcal{Y}(\boldsymbol{x})$, respectively. This challenge can be addressed with different generalizations and adaptations of the adaptive discretization-based algorithm for SIPs proposed by Blankenship and Falk [14], which is in turn inspired by [90].

Conceptually, the approach of [14] replaces the infinite index set $\mathcal{Y}$ by a finite discretized set $\mathcal{Y}^d \subsetneq \mathcal{Y}$. This discretized problem gives an approximation of (SIP). Through an adaptive refinement scheme, points are added to $\mathcal{Y}^d$, and the approximation of (SIP) is improved.

All the algorithms implemented as solvers in libDIPS are conceptually closely related to the approach of [14]. Therefore, we will give a conceptual description of the algorithms in the context of adapting the central algorithmic idea of [14]. The implemented solvers are listed in Table 2. We refer the interested reader to the original publications listed in Table 2 for a detailed description of the algorithm underlying the respective solver.

Program Class	Solver Name	Original Publication	Comment
SIP	*B&F*	[14]	Finite termination with feasible points not guaranteed
	RRHS	[78]	
	Oracle	[117]	
	Hybrid	[26]	
MINMAX	*Minmax*	[32]	
GSIP	*GSIP-RRHS*	[78]	Finite termination not guaranteed [49]
	*	[28]	Relax GSIP to derive an SIP; solve resulting SIP with any SIP solver *, c.f., Section 3.2
ESIP	*B&F*	[27]	Upper-bounding procedure of [27] not implemented
BLP	*BLP-Box*	[28]	
	BLP-noBox	[79]	

Table 2: Overview of the adaptive discretization-based algorithms implemented in libDIPS.

3.1 Approaches for SIPs

Algorithmic Approach Underlying *B&F*

The SIP algorithm of Blankenship and Falk [14] proposes to relax (SIP) by replacing the infinite index set with a finite one, i.e., $\mathcal{Y}^{LBD} \subsetneq \mathcal{Y}$. We implement this algorithm in libDIPS as the solver called *B&F*. The resulting single-level upper-level problem

$$\begin{aligned} \min_{\boldsymbol{x}\in\mathcal{X}} \quad & f(\boldsymbol{x}) \\ \text{s.t.} \quad & \boldsymbol{g}^u(\boldsymbol{x}, \boldsymbol{y}^k) \leqslant \boldsymbol{0},\ \forall\ \boldsymbol{y}^k \in \mathcal{Y}^{LBD} \\ & \mathcal{X} := \left\{\boldsymbol{x} \in \left[\boldsymbol{x}^{lb}, \boldsymbol{x}^{ub}\right] \subsetneq \mathbb{R}^{n_x} : \boldsymbol{v}^{iu}(\boldsymbol{x}) \leqslant \boldsymbol{0},\ \boldsymbol{v}^{eu}(\boldsymbol{x}) = \boldsymbol{0}\right\}, \end{aligned} \tag{LBP}$$

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

$$\begin{aligned} \max_{\boldsymbol{y}\in\mathcal{Y}} \quad & \max_{j\in\{1\dots n_{gu}\}} g_j^u(\bar{\boldsymbol{x}},\boldsymbol{y}) \\ \text{s.t.} \quad & \mathcal{Y} := \left\{\boldsymbol{y}\in\left[\boldsymbol{y}^{lb},\boldsymbol{y}^{ub}\right]\subsetneq\mathbb{R}^{n_y} : \boldsymbol{v}^{il}(\boldsymbol{y})\leqslant\boldsymbol{0},\ \boldsymbol{v}^{el}(\boldsymbol{y})=\boldsymbol{0}\right\}. \end{aligned} \tag{LLP}$$

If the optimal objective value of (LLP) is less than or equal a predefined $\varepsilon^a \geqslant 0$, $\bar{\boldsymbol{x}}$ is ε^a-SIP-feasible (c.f., Definition 4); 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., $\mathcal{Y}^{LBD} \leftarrow \mathcal{Y}^{LBD} \cup \boldsymbol{y}^*$; then, (LBP) is solved again.

Remark 1 Instead of solving a single lower-level problem, i.e., (LLP), where we maximize over all entries of $\boldsymbol{g}^u$, 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 $\boldsymbol{g}^u$, 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 $\boldsymbol{g}^u$ a separate discretization $\mathcal{Y}^{d,i}$ with $i = 1, ..., n_{gu}$.

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

It has been proven numerous times in literature [14, 58, 77] that if (SIP) is feasible and $\varepsilon^a = 0$, the accumulation points of this algorithm are optimal SIP-feasible points (c.f., Definition 6). Additionally, if (SIP) is infeasible, (LBP) will be infeasible after finitely many iterations. However, there can generally be no finite termination guarantee for $\varepsilon^a = 0$. Finite termination can only be guaranteed for $\varepsilon^a > 0$, though generating an SIP-feasible point is usually not generated in the latter case.

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 adaptions of the underlying algorithm have been proposed to generate improving sequences of upper bounds (UBD). The following algorithms have in common that, akin to [14], they iteratively solve a sequence of subproblems, i.e., 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 [77], 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 [117], implemented as *Oracle*, utilizes an oracle problem to conduct a bisection search in the objective space. The algorithm proposed by [26], implemented as *Hybrid*, attempts to combine the best of both, *RRHS* and *Oracle*.

As mentioned in the introduction, other upper-bounding approaches exist, but since they are not currently implemented in libDIPS, we refrain from reviewing them here. For this reason, we focus on the mentioned algorithms in the following:

RRHS generates upper bounds by first relaxing (SIP) by replacing the set $\mathcal{Y}$ with a set of finite cardinality, i.e., $\mathcal{Y}^{UBD} \subsetneq \mathcal{Y}$ (equivalent to *B&F*). However, the discretized (semi-infinite) constraint is restricted by the right-hand side with a restriction parameter $\varepsilon^r > 0$, which is initialized with $\varepsilon^{r0} > 0$, to

$$\begin{aligned} \min_{\boldsymbol{x}\in\mathcal{X}} \quad & f(\boldsymbol{x}) \\ \text{s.t.} \quad & \boldsymbol{g}^u\left(\boldsymbol{x},\boldsymbol{y}^k\right) \leqslant -\varepsilon^r\cdot\boldsymbol{1},\ \forall\ \boldsymbol{y}^k\in\mathcal{Y}^{UBD} \\ & \mathcal{X} := \left\{\boldsymbol{x}\in\left[\boldsymbol{x}^{lb},\boldsymbol{x}^{ub}\right]\subsetneq\mathbb{R}^{n_x} : \boldsymbol{v}^{iu}(\boldsymbol{x})\leqslant\boldsymbol{0},\ \boldsymbol{v}^{eu}(\boldsymbol{x})=\boldsymbol{0}\right\}. \end{aligned} \tag{UBP}$$

Similar to *B&F*, (UBP) and subsequently (LLP) are iteratively solved to determine SIP-feasibility of the candidate points generated by (UBP) and the solution points of (LLP) are used to populate $\mathcal{Y}^{UBD}$. Note that neither a relaxation nor a restriction of (SIP) is generally attained by (UBP) through the discretization combined with the restriction. However, finite termination is guaranteed through a decreasing rule for ε^r as the algorithm proceeds, i.e., $\varepsilon^r \leftarrow \frac{\varepsilon^r}{\varepsilon^{red}}$, which is applied under certain conditions; an increasingly dense discretization $\mathcal{Y}^{UBD}$; and further suitable assumptions (existence of an ε^f-optimal SIP-Slater point, c.f., Definition 2; global solution of the subproblems; continuity of all functions; and compact sets).

Note that the performance of the upper-bounding procedure is strongly influenced by the initial restriction parameter ε^{r0}, the rate of reduction ε^{red}, and the initial discretization $\mathcal{Y}^{UBD}$. For example, a (too) fast reduction of the restriction parameter, or a (too) large ε^{red}, can lead to many iterations [26].

Oracle uses an oracle adaption of [14] to determine if a target objective value f^t is attainable. The adapted discretized upper-level problem is given as

$$\begin{aligned} \min_{\boldsymbol{x}\in\mathcal{X}} \quad & \max\left\{ f(\boldsymbol{x}) - f^t, \max_{\substack{\boldsymbol{y}^k\in\mathcal{Y}^{ORA},\\ j\in\{1\dots n_{gu}\}}} g_j^u(\boldsymbol{x},\boldsymbol{y}^k) \right\} \\ \text{s.t.} \quad & \mathcal{X} := \left\{\boldsymbol{x}\in\left[\boldsymbol{x}^{lb},\boldsymbol{x}^{ub}\right]\subsetneq\mathbb{R}^{n_x} : \boldsymbol{v}^{iu}(\boldsymbol{x})\leqslant\boldsymbol{0},\ \boldsymbol{v}^{eu}(\boldsymbol{x})=\boldsymbol{0}\right\}, \end{aligned} \tag{ORA}$$

with $\mathcal{Y}^{ORA}\subsetneq\mathcal{Y}$. This problem can be reformulated to avoid the max-operator, c.f., (ORA-REF).

Given preexisting upper and lower bounds on the optimal objective value of (SIP), a bisection search in the objective space is conducted. 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 infeasible, the optimal solution point of (LLP) is used for the discretization of $\mathcal{Y}^{ORA}$. Else $\bar{\boldsymbol{x}}$ is 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*.

Hybrid aims to combine the ideas of the two previous approaches. Recall that the performance of *RRHS* depends on the hyperparameters ε^{r0} and ε^{red}. *Hybrid* essentially uses the same iterative approach for the upper- and lower-bounding as *RRHS*. Additionally, by solving

$$\begin{aligned} \min_{\boldsymbol{x}\in\mathcal{X},\eta} \quad & -\eta \\ \text{s.t.} \quad & f(\boldsymbol{x}) - f^{RES} \leqslant 0 \\ & \boldsymbol{g}^u(\boldsymbol{x},\boldsymbol{y}^k) \leqslant -\eta\cdot\boldsymbol{1},\ \forall\ \boldsymbol{y}^k\in\mathcal{Y}^{RES} \\ & \mathcal{X} := \left\{\boldsymbol{x}\in\left[\boldsymbol{x}^{lb},\boldsymbol{x}^{ub}\right]\subsetneq\mathbb{R}^{n_x} : \boldsymbol{v}^{iu}(\boldsymbol{x})\leqslant\boldsymbol{0},\ \boldsymbol{v}^{eu}(\boldsymbol{x})=\boldsymbol{0}\right\}, \end{aligned} \tag{RES}$$

a modified version of the subproblem (ORA), an adaptive (optimal) update to the restriction parameter ε^r can be generated in some of the iterations. Using this updated restriction parameter in (UBP), *Hybrid* aims to eradicate the performance dependence on the hyperparameters ε^{r0} and ε^{red}.

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 program classes GSIP, ESIP, MINMAX, and BLP, covered in the following sections, are also listed in Table 3.

Program 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*	(LBP), (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: List of subproblems used in the solvers. Note that the solvers automatically generate some auxiliary problems.

Algorithmic Approach Underlying *Minmax*

In [32], a specialization of the approach from *B&F* 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 given by the lower-level problem

$$\max_{y \in \mathcal{Y}} f(\bar{\boldsymbol{x}}, \boldsymbol{y}).$$

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.

3.2 Approaches for GSIPs

In literature, two main approaches exist for the global solution of (GSIP) in the context of adaptive discretization-based methods: (i) relaxation of (GSIP) to an SIP and the solution thereof with an SIP approach and (ii) the adaption of the underlying approach used in *RRHS* to GSIPs.

GSIP to SIP Reformulation and Relaxation

A common approach is to reformulate and relax (GSIP) to an SIP. By defining

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

with $i \in \{1 \dots n_{gu}\}$ and inserting it as the semi-infinite constraint in (SIP), an SIP relaxation of (GSIP) is obtained [25]. As shown in [43], this relaxation is under certain assumptions equivalent to relaxing the feasible set of (GSIP) to its closure. Hence, we assume that the minimum of the relaxed problem is equal to the infimum of the original problem [25, 78]. If this property is not present, the relaxation can be strict. State-of-the-art SIP solvers can be applied, as described in Section 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 Section 3 whenever a derived SIP is solved with an SIP solver.

Adaption of *RRHS* for GSIPs

The previous approach, i.e., the solution of the derived SIP with SIP solvers, has the inherent potential disadvantage that the original GSIP structure is ignored. On the contrary, the algorithm published by [78], implemented as *GSIP-RRHS*, considers the GSIP structure. Similar to *RRHS*, the authors in [78] employ a restriction of the right-hand side approach to generate upper bounds and an adaption of *B&F* to generate lower bounds. [78] 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

$$\begin{aligned} g^{u,*} = \max_{\boldsymbol{y}\in\mathcal{Y}} \quad & \max_{j\in\{1\dots n_{gu}\}} g_j^u(\bar{\boldsymbol{x}},\boldsymbol{y}) \\ \text{s.t.} \quad & \boldsymbol{g}^l(\boldsymbol{x},\boldsymbol{y}) \leqslant \boldsymbol{0} \\ & \mathcal{Y} \coloneqq \left\{\boldsymbol{y}\in\left[\boldsymbol{y}^{lb},\boldsymbol{y}^{ub}\right]\subsetneq\mathbb{R}^{n_y} : \boldsymbol{v}^{il}(\boldsymbol{y})\leqslant\boldsymbol{0},\ \boldsymbol{v}^{el}(\boldsymbol{y})=\boldsymbol{0}\right\} \end{aligned} \tag{GSIP-LLP}$$

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

$$\begin{aligned} \min_{\boldsymbol{y}\in\mathcal{Y}} \quad & \max_{j\in\{1\dots n_{gl}\}} g^l(\bar{\boldsymbol{x}},\boldsymbol{y}) \\ \text{s.t.} \quad & \max_{j\in\{1\dots n_{gu}\}} g_j^u(\bar{\boldsymbol{x}},\boldsymbol{y}) \geqslant \alpha\cdot g^{u,*} \\ & \mathcal{Y} \coloneqq \left\{\boldsymbol{y}\in\left[\boldsymbol{y}^{lb},\boldsymbol{y}^{ub}\right]\subsetneq\mathbb{R}^{n_y} : \boldsymbol{v}^{il}(\boldsymbol{y})\leqslant\boldsymbol{0},\ \boldsymbol{v}^{el}(\boldsymbol{y})=\boldsymbol{0}\right\} \end{aligned} \tag{GSIP-AUX}$$

with $\alpha > 0$ is solved to find a GSIP-LLP-Slater point, c.f., Definition 3. This GSIP-LLP-Slater point is needed because only a discretization point that strictly fulfills the coupling inequality constraints $\boldsymbol{g}^l(\boldsymbol{x},\boldsymbol{y}) \leqslant \boldsymbol{0}$ will, if added to the discretization, provide a restriction in the derived (LBP). The solution point of (GSIP-LLP) will not necessarily provide a restriction if used as a discretization point.

The original publication searched for a GSIP-LLP-Slater point whose relative sub-optimality in the lower level was bounded over all iterations by a fixed factor $\alpha > 0$. However, depending on the a-priory chosen α, generating a GSIP-LLP-Slater point might fail [49]. The implementation of this approach, i.e., *GSIP-RRHS*, tries to address the issues raised by [49] by adaptively choosing α whenever a GSIP-LLP-Slater point is not attained by (GSIP-AUX), c.f., Section 4.2. However, in certain cases, numerical issues connected to identifying GSIP-LLP Slater points persist for *GSIP-RRHS*, c.f., Section 6.2.

3.3 Approaches for ESIPs

The algorithmic extension to ESIPs, proposed in [27], 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

$$\begin{aligned} \max_{\boldsymbol{y}\in\mathcal{Y}}\min_{\boldsymbol{z}\in\mathcal{Z}} \quad & \max_{j\in\{1\dots n_{gu}\}} g_j^u(\bar{\boldsymbol{x}},\boldsymbol{y},\boldsymbol{z}) \\ \text{s.t.} \quad & \mathcal{Y} \coloneqq \left\{\boldsymbol{y}\in\left[\boldsymbol{y}^{lb},\boldsymbol{y}^{ub}\right]\subsetneq\mathbb{R}^{n_y} : \boldsymbol{v}^{il}(\boldsymbol{y})\leqslant\boldsymbol{0},\ \boldsymbol{v}^{el}(\boldsymbol{y})=\boldsymbol{0}\right\} \end{aligned} \tag{ESIP-MINMAX}$$

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

$$\begin{aligned} \min_{\boldsymbol{x}\in\mathcal{X},\boldsymbol{z}^1\in\mathcal{Z}\ldots\boldsymbol{z}^{|\mathcal{Y}^{LBD}|}\in\mathcal{Z}} \quad & f(\boldsymbol{x}) \\ \text{s.t.} \quad & \boldsymbol{g}^u\left(\boldsymbol{x},\boldsymbol{y}^k,\boldsymbol{z}^k\right) \leqslant \boldsymbol{0},\ \forall\ k\in\left\{1\ldots\left|\mathcal{Y}^{LBD}\right|\right\} \\ & \mathcal{X} \coloneqq \left\{\boldsymbol{x}\in\left[\boldsymbol{x}^{lb},\boldsymbol{x}^{ub}\right]\subsetneq\mathbb{R}^{n_x} : \boldsymbol{v}^{iu}(\boldsymbol{x})\leqslant\boldsymbol{0},\ \boldsymbol{v}^{eu}(\boldsymbol{x})=\boldsymbol{0}\right\}, \end{aligned} \tag{ESIP-LBP}$$

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

3.4 Approaches for BLPs

For BLPs we implement two solvers, namely ***BLP-Box*** based on [79] and ***BLP-noBox*** based on [28]. For both, the connection to the approach of [14] is based on the reformulation of the BLP to a GSIP using the value function reformulation

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

Accordingly, both approaches utilize

$$\begin{aligned} \min_{\boldsymbol{x}\in\mathcal{X},\boldsymbol{y}\in\mathcal{Y}(\boldsymbol{x})} \quad & f(\boldsymbol{x},\boldsymbol{y}) \\ \text{s.t.} \quad & \boldsymbol{g}^u(\boldsymbol{x},\boldsymbol{y})\leqslant\boldsymbol{0} \\ & \boldsymbol{x}\in\tilde{\mathcal{X}}\left(\boldsymbol{z}^k\right) \implies h(\boldsymbol{x},\boldsymbol{y})-h\left(\boldsymbol{x},\boldsymbol{z}^k\right)\leqslant 0,\ \forall\ \boldsymbol{z}^k\in\mathcal{Y}^{LBD} \\ & \mathcal{X} \coloneqq \left\{\boldsymbol{x}\in\left[\boldsymbol{x}^{lb},\boldsymbol{x}^{ub}\right]\subsetneq\mathbb{R}^{n_x} : \boldsymbol{v}^{iu}(\boldsymbol{x})\leqslant\boldsymbol{0},\ \boldsymbol{v}^{eu}(\boldsymbol{x})=\boldsymbol{0}\right\}, \end{aligned} \tag{BLP-LBP}$$

to generate a LBD based on the discretization $\mathcal{Y}^{LBD}$. The main difference is how the set $\tilde{\mathcal{X}}$ is chosen. In [79], the authors compute boxes in each iteration k around the current iterate for the upper-level variables $\bar{\boldsymbol{x}}$ such that within these boxes, it can be assured that the discretization point $\boldsymbol{z}^k$ stays feasible concerning the lower-level constraints. This approach is implemented in our solver *BLP-Box*. The approach in [28] generates the set $\tilde{\mathcal{X}}$ in a parametric way by setting $\tilde{\mathcal{X}}(\boldsymbol{z}) \coloneqq \{\boldsymbol{x}\in\mathcal{X} : \boldsymbol{z}\in\mathcal{Y}(\boldsymbol{x})\}$. To generate discretization points, a Slater point of the lower level with respect to the coupling constraint $\boldsymbol{g}^l$ 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

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

with the estimate $\bar{h}$ for the optimal value of the lower level at the current value for the upper-level variables $\bar{\boldsymbol{x}}$, a small positive tolerance $\varepsilon^{l,UBD}$, and a known lower bound on the upper-level objective LBD.

4 libDIPS – Discretization-Based Semi-Infinite and Bilevel Programming Solvers

As motivated in the introduction and highlighted in Section 3, the reviewed algorithms and implemented solvers are closely related and, hence, can benefit from a shared infrastructure. By combining all these solvers in one software, we can compare them

more fairly because they use the same programming language, use a single common code-base, and access subsolvers through the same interface. Therefore, we implemented all solvers in the open-source C++ software libDIPS – Discretization-Based Semi-Infinite and Bilevel Programming Solvers, already used in a preliminary form in [25].

The latest version of libDIPS, including the text-based parser, support for the MPI parallelized version of MAiNGO [15], 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`, and has been tested under Microsoft Windows 10 and Linux. Further information on downloading and compiling libDIPS and its dependencies can be found on the documentation page on the libDIPS repository.

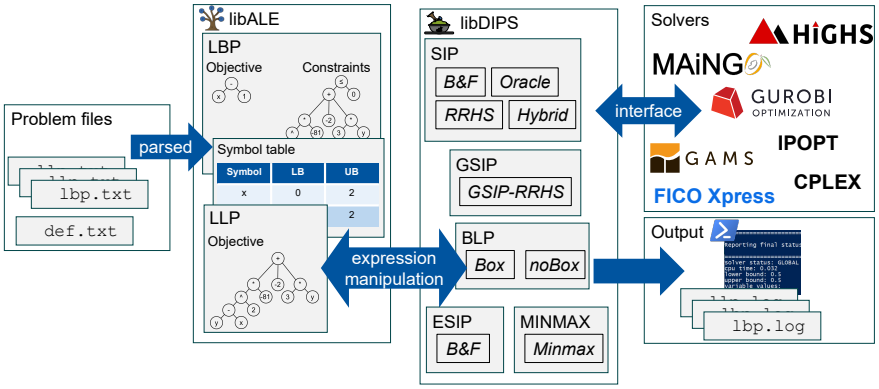


Fig. 1: Software structure overview.

The general structure of the software is depicted in Figure 1. Based on this structure, we summarize the following features of the software

- Problem description: Subproblems are provided by the user in an easily editable and human-readable form using the domain-specific language provided by libALE [134].
- libALE is used for reading the user input (subproblems), which then internally stores the objective and constraints of each subproblem as logical expression trees. The subproblems are represented symbolically using libALE. Variable bounds and parameter values are tracked with a symbol table.
- The solvers listed in Table 2 are implemented in libDIPS. A framework for discretization-based solvers allows for easy extension of the library with additional solvers. For example, necessary symbol and expression manipulations of the subproblems needed by the solvers are carried out in libALE. This allows us to robustly build the subproblems needed in the algorithms.
- libDIPS interfaces to several state-of-the-art optimization solvers, i.e., CPLEX [56], GUROBI [46], IPOPT [122], FICO Xpress [31], and MAiNGO [15] as well as the GAMS C++ API [36]. This enables the user to switch between them easily. Numerous additional solvers are accessible via GAMS, e.g., BARON [96], ANTIGONE [75]. The modular structure allows the user to implement interfaces to additional subsolvers as needed.
- Output is printed onto the command line, and additional log files are written.

There are two options for formulating a hierarchical optimization problem for the 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

```
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 exemplary problem (Ex). Note that `lbp_y_disc` represents $\mathcal{Y}^{LBD}$ and is initialized as an empty set. `lbp_k_disc` corresponds to the indices of iterations and is internally updated by the used solver.

provide templated input files that allow deriving the necessary subproblems automatically from a high-level problem description. In the following section, we elaborate on the benefits of individually formulating the subproblems with a small example.

4.1 Exemplary Solution of an SIP via *B&F*

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

$$\begin{aligned} \min_{\boldsymbol{x}\in\mathcal{X}:=[-3,3]^2} \quad & f(\boldsymbol{x}) := 2x_1^2 + x_2^2 \\ \text{s.t.} \quad & g^u(\boldsymbol{x}, y) := \min\{x_1 + y_1, x_2 + y_2\} \leqslant 0, \ \forall\ y \in \mathcal{Y} \\ & v^{eu}(\boldsymbol{x}) := x_1 + x_2 - 1 = 0 \\ & \mathcal{Y} = [-1, 1]^2 \end{aligned} \tag{Ex}$$

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 primarily intuitive. For a detailed introduction to the domain-specific language provided by libALE and the syntax on how to define an optimization problem, the reader may refer to the documentation of libALE [134] and MAiNGO [15].

If we were to naively or automatically generate the subproblem (LLP) from the problem description (i.e. from f, g^u, v^{eu} and $\mathcal{Y}$), the resulting objective would read $\max_{\boldsymbol{y}\in\mathcal{Y}} \min\{x_1 + y_1, x_2 + y_2\}$. 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 $\max_{\boldsymbol{y}\in\mathcal{Y}} t$ s.t. $t \leqslant x_1 + y_1,\ t \leqslant x_2 + y_2$. This formulation allows us to solve a linear optimization problem instead of 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 meaning of some of the used symbols in the subproblems. For *B&F*, the user must provide the algorithm with the lower-level variable

```
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*.

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 input example files for the other solvers are given in the libDIPS repository.

4.2 Implementation changes of the original algorithms in libDIPS

As revisited in Section 2, the implemented algorithms are conceptually closely related. Hence, we were able to enhance the implemented solvers by utilizing the 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 in the Supporting Information shows a detailed overview of the improvements of the implemented algorithms in libDIPS.

5 Benchmark Test Set

As mentioned in the introduction, one of the main goals of this manuscript is to compare the different algorithms on a more comprehensive array of benchmark problem instances. For this task, we collected problem instances from the literature of the different program classes. All problem instances are available in the libDIPS repository. The vast majority of the collected problem instances are academic examples that were used for the algorithm presentation by the respective authors. With libDIPS, we provide easy-to-use software for the solution of hierarchical programs with which real-life problem instances can be implemented and, in the future, be incorporated into the

benchmark test set. Nonetheless, these academic examples contain numerous challenging problems that were specifically chosen by the respective authors to highlight and pinpoint certain problems in existing/previous algorithms.

We did not exploit individual opinionated problem formulations within the benchmark test set, i.e., we tried to keep the problem formulations as close as possible to the initially published problem formulation. 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 program 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 choosen as it is preferable for the used subsolvers.

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

We performed a two-step heuristic procedure to avoid duplicate problem instances in the presented benchmark test set. 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 BLP test set, we automatically checked for symbolic equivalence of the objective functions to narrow down the number of potential duplicates in the first step. For similar problem instances, e.g., the problem instances differed by an additional constraint, we added both problem instances to the benchmark test set and a remark in the corresponding `def.txt` files. Both problem instances were added, even if the constraint did not change the optimal solution, as equivalence is not straightforward to derive from the problem statement, and these additional constraints might influence the performance of the used subsolver. We proceeded in the same manner when an original problem instance was cited, but the presented problem instance was different.

For SIPs, we started with the existing benchmark test sets from [77, 120, 124]. Additionally, we added problem instances from numerous publications. Table 4 shows a complete list of the used publications. Note that in Table 4 and in all following listings, only the oldest found publication with the implemented problem instance is listed. After removing duplicates, the benchmark test set contains 294 SIPs; for a complete list of implemented problem instances, see Table S2 in Section S5 of the Supporting Information.

For MINMAX, we collected problem instances from numerous publications. After removing duplicates, the benchmark test set 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 benchmark test set of [78] and added problem instances from GSIPLib&Gen [99]. The problem instances taken from [99] 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 [80] and combined them with the existing benchmark test set BASLib [84] and BOLIB v2 [133]. For some of the problem instances, we observed that there are constraints implemented as part of the lower-level feasible set $\mathcal{Y}(\boldsymbol{x})$ 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 behind this transformation is that these constraints can be trivially detected, and without the transformation, the solvers encountered numerical issues or had 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.

Program Class	Used publications (oldest found)
SIP	[1, 6, 8, 10, 12–14, 16, 17, 21, 22, 24, 33, 35, 37, 38, 40, 44, 45, 47–49, 54, 55, 62, 64–66, 68–72, 74, 76, 80–82, 85–89, 94, 95, 97, 98, 102, 104, 108, 110, 111, 113, 114, 116, 117, 120, 123, 124, 128–132]
GSIP	[6, 19, 24, 49, 50, 57, 59, 64, 67, 69, 78, 91–93, 97, 99, 100, 103, 104, 106, 107, 109, 115, 118, 121]
BLP	[79, 84, 133]
MINMAX	[2, 3, 7, 23, 24, 30, 34, 37, 39, 44, 51–53, 55, 74, 82, 130, 132]

Table 4: List of publications used in the benchmark test set.

We categorized all problem instances according to the program 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

The composition of the benchmark test set is shown in Table 5.

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 performed on a single core with an Intel Xeon Platinum 8160 Processor "SkyLake" running at 2.1 GHz. All subproblems are solved using MAiNGO version 0.7.1. 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 to 1×10^{-9}. Note that for some automatically

UL \ LL	LP	QP	QCQP	NLP
LP	22	22	1	75
QP	1	5	2	21
QCQP	2	12	5	11
NLP	9	3	3	100

(a) SIP benchmark test set.

UL \ LL	LP	QP	QCQP	NLP
LP	11	0	4	4
QP	8	3	6	1
QCQP	6	7	8	8
NLP	1	0	4	15

(b) GSIP benchmark test set.

UL \ LL	LP	QP	QCQP	NLP
LP	0	1	0	1
QP	0	0	0	0
QCQP	0	13	0	44
NLP	0	3	0	21

(c) MINMAX benchmark test set.

UL \ LL	LP	QP	QCQP	NLP
LP	31	0	0	0
QP	7	0	0	0
QCQP	18	41	7	0
NLP	2	9	10	42

(d) BLP benchmark test set.

Table 5: Composition of the benchmark test sets for the different problem categories. UL = problem class of the upper level, specifically the respective LBD with a given discretization, LL = problem class of the lower level with fixed upper-level variables.

generated subproblems, we use specific fixed settings to speed up the convergence of MAiNGO, e.g., in order 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, the optimality tolerances used can be configured by the user. 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 the output settings have been left at the default values.

We use each program class's problem instances of the benchmark test set presented in Section 5. To compare the performance of the individual solvers within a program class, we utilize Dolan-Moré plots; we plot the percentage of problem instances solved over the CPU time or the time factor defined as

$$\textit{Time factor} = \frac{\textit{CPU time of problem instance } i}{\textit{CPU time of problem instance } i \textit{ of the fastest solver}}.$$

We mainly discuss the time factor plots and, thus, relative performance. Our motivation is to lessen the importance of the size of the problem instances. However, for problems with very small solution times, minor absolute time differences can cause a large difference in terms of the time factor. To combat this, all CPU times under 1 seconds have been rounded up to 1 seconds as we can not fairly differentiate relative run-time measurements for run times under 1 second.

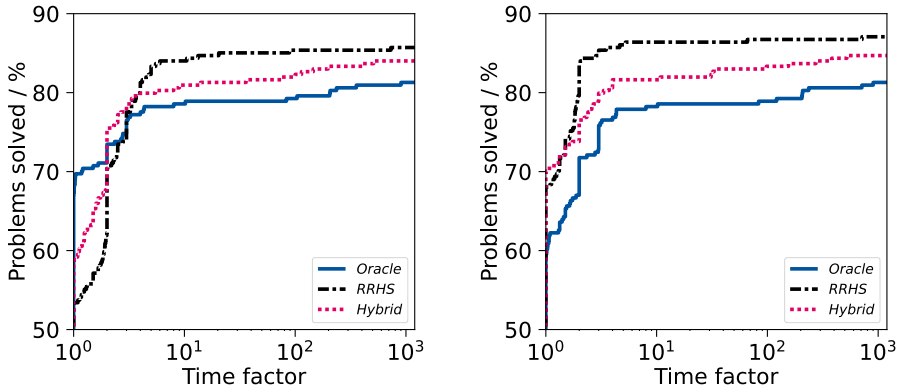
We briefly investigated the consistency of the measured run times by measuring three repeated runs of *RRHS* on the SIP benchmark test set. On average, the standard deviation over the three repeated runs, normalized by the mean, is 0.02 and the maximum of this normalized standard deviation is 0.16. Since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

As *B&F* terminates with ε^a-feasible points, we exclude *B&F* from the performance tests.

6.1 Performance Results of SIP solvers

6.1.1 "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 hyperparameters of each solver are set to the values of their respective original publication, c.f., Table 6 (for CPU time plot see Figure S1a in Section S4.1 of the Supporting Information). Quantitatively, the performance of the solvers on the more extensive benchmark test set is similar to the results published in [25]: *Oracle* is the fastest for the largest fractions of problems, but when it is not the fastest, it is often significantly slower and is the least robust. In contrast, *RRHS* is the most robust and can solve the most problems within a time factor of 10, but is the fastest solver in the least amount of problems. The solver *Hybrid* is able to solve significantly more problems than *Oracle* and outperforms the *RRHS* solver 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 hyperparameter 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*.



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

(b) Solvers with tuned hyperparameters.

Fig. 2: Time factor performance plots of SIP solvers: *RRHS*, *Oracle*, and *Hybrid*. Without tuning, a clear trade-off between fraction of problem solved overall and fraction of problem solved in the shortest exists between the solvers. With tuning *RRHS* performs best.

6.1.2 Hyperparameter tuning for *RRHS* and *Hybrid*

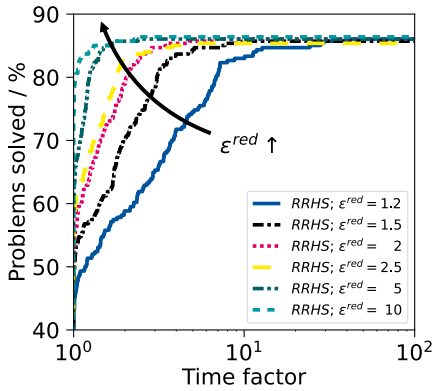
Recall that the performance of the upper-bounding procedure, and hence overall performance, of *RRHS* and *Hybrid* is influenced by the initial restriction parameter ε^{r0}; the rate of reduction, i.e., ε^{red}; and the initial discretization. Therefore, we performed a hyperparameter study for *RRHS* and *Hybrid* varying the two hyperparameters $\varepsilon^{r0} = \{0.1, 1, 2, 5, 8, 10\}$ and $\varepsilon^{red} = \{1.2, 1.5, 2, 2.5, 5, 10\}$. The results of the parameter study are shown in Figures 3 and 4 (for the CPU time performance plots see Figure S1b in Section S4.1 of the Supporting Information).

Program Class	Algorithm Solver Name	Hyperparemeters in original publication	Tuned hyperparameters
SIP	*RRHS*	$\varepsilon^{r0} = 1$, $\varepsilon^{red} = 1.5$	$\varepsilon^{r0} = 0.1$, $\varepsilon^{red} = 10$
	Oracle	–	–
	Hybrid	$\varepsilon^{r0} = 1$, $\varepsilon^{red} = 1.2$	$\varepsilon^{r0} = 0.1$, $\varepsilon^{red} = 10$
GSIP	*GSIP-RRHS*	$\varepsilon^{r0} = 1$, $\varepsilon^{red} = 2$, $\alpha^0 = 0.5$, $\alpha^{red} = *$	$\varepsilon^{r0} = 1$, $\varepsilon^{red} = 2$, $\alpha^0 = 0.25$, $\alpha^{red} = 1.2$
	RRHS	–	$\varepsilon^{r0} = 5$, $\varepsilon^{red} = 5$
	Oracle	–	–
	Hybrid	–	$\varepsilon^{r0} = 0.1$, $\varepsilon^{red} = 5$

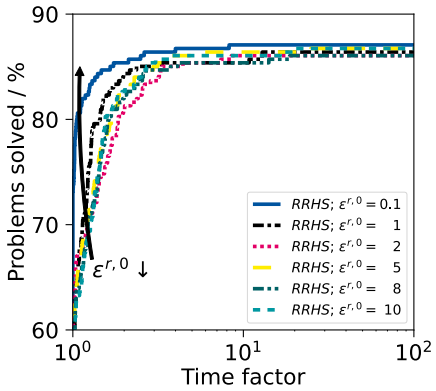
Table 6: Hyperparameter values used in the original publications [26, 77, 78, 117] and tuned hyperparameter values. *GSIP-RRHS*: Note that [78] introduces restriction and reduction parameters that may diff between $\boldsymbol{g}^u$ and for $\boldsymbol{g}^l$; however [78] uses the same values for both. $*$: α^{red} is introduced in this work.

RRHS, for a fixed $\varepsilon^{r0} = 1$ (value in original publication [77]), performs best with a fast reduction of ε^r, c.f., Figure 3a. Furthermore, a small initial restriction ε^{r0} is beneficial, c.f., Figure 3b. Hence, choosing a big value for ε^{red} and a small ε^{r0} seems beneficial for the considered benchmark test set. With tuned hyperparameters, i.e., $\varepsilon^{red} = 10$ and $\varepsilon^{r0} = 0.1$, *RRHS* was able to solve 87 % 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 7.

Hybrid, for a fixed $\varepsilon^{r0} = 1$ (value in original publication [26]), performs best with a fast reduction of ε^r, c.f., Figure 4a. Furthermore, a small initial restriction ε^{r0} is beneficial, c.f., Figure 4b. With tuned hyperparameters, i.e., $\varepsilon^{red} = 10$ and $\varepsilon^{r0} = 0.1$, *Hybrid* was able to solve 85 % 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 7.

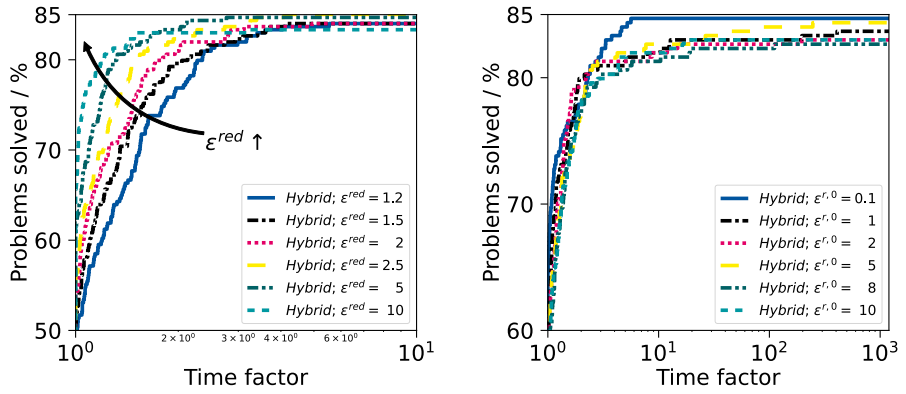


(a) *RRHS*: $\varepsilon^{r0} = 1$ and $\varepsilon^{red} = \{1.2, 1.5, 2, 2.5, 5, 10\}$.



(b) *RRHS*: $\varepsilon^{r0} = \{0.1, 1, 2, 5, 8, 10\}$ and $\varepsilon^{red} = 10$.

Fig. 3: Time factor performance plots of SIP hyperparameter study for *RRHS*. Performance improves with increasing ε^{red} and decreasing ε^{r0}.



(a) *Hybrid*: $\varepsilon^{r0} = 1$ and $\varepsilon^{red} = \{1.2, 1.5, 2, 2.5, 5, 10\}$.

(b) *Hybrid*: $\varepsilon^{r0} = \{0.1, 1, 2, 5, 8, 10\}$ and $\varepsilon^{red} = 10$.

Fig. 4: Time factor performance plots of SIP hyperparameter study for *Hybrid*. Performance improves with increasing ε^{red}. No systematic behavior is seen for ε^{r0}.

Figure 2b shows that after hyperparameter tuning *RRHS* outperforms the other solvers. 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 hyperparameters. This is most likely 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* hyperparameters are chosen. In summary, the performance of *RRHS* is superior compared to the other solvers, especially if parameter tuning can be performed.

6.1.3 Performance Result of *Minmax*

Figure 5 shows the performance of *Minmax* on the benchmark test set. We refrained from a comparison to SIP solvers using the reformulation given in MINMAX-REF as the specialized algorithm is clearly preferable: The specialized algorithm gives 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 7.

6.2 Performance Results of GSIP algorithms

We conducted a similar hyperparameter study for the GSIP solvers as for the SIP solvers. The problem instances of the GSIP benchmark study were solved with *GSIP-RRHS* and the corresponding derived SIPs, c.f., Section 3.2, with the SIP solvers *Oracle*, *RRHS*, and *Hybrid*. We varied the hyperparameters $\varepsilon^{r0} = \{0.1, 1, 5\}$, $\varepsilon^{red} = \{1.2, 2, 5\}$, $\alpha^0 = \{0.25, 0.5, 0.95\}$, and $\alpha^{red} = \{1.2, 2, 5\}$ of the algorithms *GSIP-RRHS*, *RRHS*, and *Hybrid*, respectively. Figure 6 clearly shows that *GSIP-RRHS* performs best for many problems for tuned hyperparameters (for CPU time, subproblem

Program Class	Algorithm Solver Name	non-solved / total	exceeded max. time	Subsolver errors	Algorithmic errors
SIP	*RRHS*	38/294	7	18	13
	Oracle	48/294	10	35	3
	Hybrid	45/294	8	24	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	*B&F*	20/ 83	–	20	–

Table 7: Number of errors encountered in the benchmark study using tuned hyperparameters, c.f., Table 6. The subsolver errors category includes problems instances where MAiNGO did not solve a subproblem globally or an exception was thrown by MAiNGO . The algorithmic errors category includes problems 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

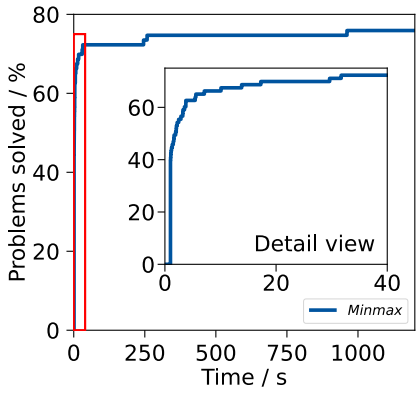


Fig. 5: CPU time performance plot of *Minmax*. Most problems are solved within 40 seconds.

number, and subproblem factor performance plots see Figures S3 and S4 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 S5 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 subsolvers. However, it remains to be seen whether an improved opinionated subproblem formulation of (LLP) could alter the 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. This is connected to the issue raised by [49], which we try to address by reducing α iteratively whenever a GSIP-LLP-Slater point is not attained by (GSIP-AUX), c.f., Sections 3.2 and 4.2. 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 7.

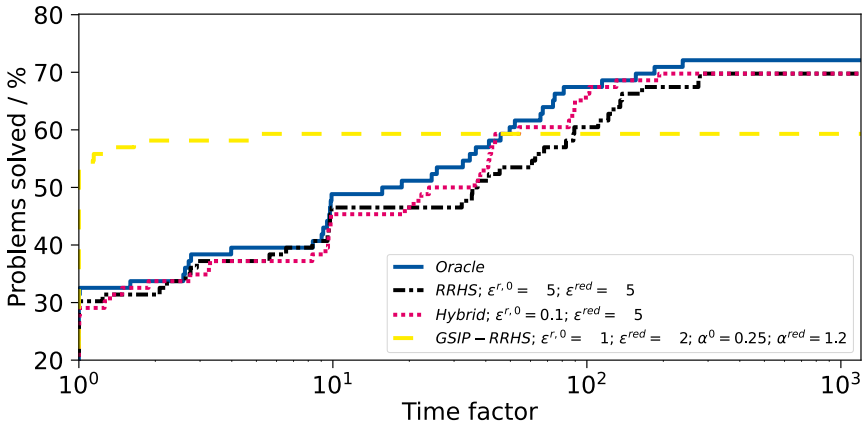


Fig. 6: Time factor performance plots of GSIP hyperparameter study for *GSIP-RRHS*, *Oracle*, *RRHS*, *Hybrid* using tuned hyperparameters according to Table 6. *GSIP-RRHS* is fastest most often, but solves less problems within the time-limit.

6.3 Performance Results of BLP algorithms

We conducted a similar hyperparameter tuning study for the BLP solvers. For *BLP-Box*, we varied $d^0 = \{0.1, 0.5, 1\}$ and $d^{red} = \{0.1, 0.5, 0.9\}$. *BLP-noBox* performs best, c.f., Figure 7. Note that a large part of *BLP-Box*'s performance disadvantage is due to subsolver errors, c.f., Table 7. The *best* hyperparameters for *BLP-Box* are $d^0 = 1$ and $d^{red} = 0.9$, while the influence of d^{red} is less significant. For a detailed listing of the reasons why some of the problem instances were not solved, refer to Table 7. We conclude that the approach of choosing $\tilde{\mathcal{X}}$ of [28] is superior to [79].

6.4 Testing of Two Exemplary Scientific Hypotheses

The extensive benchmark test set in combination with libDIPS brings the advantage that we can easily test scientific hypotheses for the algorithms. The following sections briefly describe two exemplary hypotheses to further speed up convergence, which were implemented in libDIPS and tested on the afore-introduced benchmark test set.

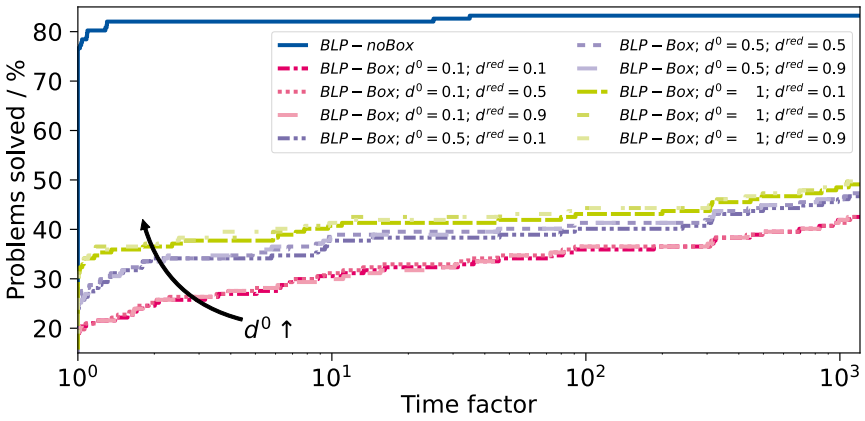


Fig. 7: Time factor performance plots of BLP hyperparameter study for *BLP-noBox* and *BLP-Box* with $d^0 = \{0.1, 0.5, 1\}$ and $d^{red} = \{0.1, 0.5, 0.9\}$. *BLP-noBox* outperforms BLP-Box for all tested parameter values. Lines with the same color correspond to the same value of the parameter d^{red}. Changes in d^{red} seem to make little difference while increasing d^0 improves performance.

6.4.1 Introduction of a Guard for the Upper-bounding Procedure in *RRHS*

RRHS, with tuned hyperparameters ($\varepsilon^{r0} = 0.1$, $\varepsilon^{red} = 10$), spends about 24 % of the total CPU time for the upper-bounding procedure over all solved problem instances (c.f, Figure S2 in Section S4.1 of the Supporting Information). The upper-bounding problem of *RRHS*, (UBP), 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 a feasible solution and thus a "good" upper bound. Hence, if ε^{r0} is a small value and we start with an empty discretization $\mathcal{Y}^{UBD} = \varnothing$, 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 $\varepsilon^a > 0$. Here, we choose $\varepsilon^a = 0.01$ and refer to this introduced guard parameter as $g^{UBP} = 0.01$. Whenever the upper-bounding procedure is skipped, the discretization of the $\mathcal{Y}^{UBD}$ is populated with the discretization point used in the lower-bounding procedure. Hence, the upper-bounding procedure is skipped until the discretization of $\mathcal{Y}^{UBD}$ has reached a sufficient density (in combination with ε^r). Figure 8 shows that this heuristic benefits problems where the solver was already competitive with a small time factor and does not negatively affect performance on the other problems (for subproblem number and subproblem factor performance plots see Figure S6 in Section S4.3 of the Supporting Information). Note that this heuristic does not impede the convergence guarantees.

6.4.2 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

$$LBD \leqslant f(x) \leqslant UBD, \tag{1}$$

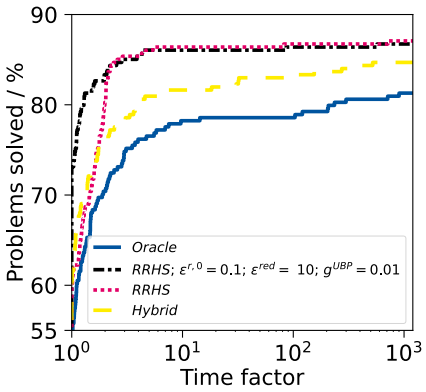


Fig. 8: CPU time factor performance plot for introduced guard parameter as $g^{UBP} = 0.01$ in *RRHS*. Comparison with tuned hyperparameters for the respective solvers. For small time-factors the guard significantly improves performance.

where LBD and UBD are the current upper and lower bound. 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) is will become infeasible. Whenever (UBP) is infeasible, ε^r is deemed "too" big; it will be reduced, and the subsequent (unnecessary) solution of (LLP) is skipped. Exemplary, we test our hypothesis on *RRHS* with tuned hyperparameters, c.f, Table 6. However, while the results shown in Figure 9 indicate that this change can have a large positive impact for a given problem instance, we did not see an overall improvement across the benchmark. The bracketing of the objective function did not reduce the number of subproblems solved (for subproblem number and factor performance plots see Figure S7 in Section S4.3 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.

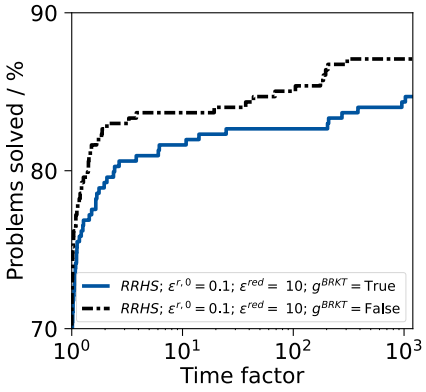


Fig. 9: Time factor performance plot comparing the approach using bracketing (g^{BRKT}=True) against the default of not using it (g^{BRKT}=False). Comparison with tuned hyperparameters of *RRHS*. Bracketing does not improve performance.

7 Conclusion and outlook

We presented libDIPS, an open-source software for adaptive discretization-based algorithms for (G)SIPs, BLPs, ESIPs, and MINMAX programs, which provides solvers based on the algorithms proposed and further developed by [14, 26–28, 78, 79, 117]. We optimized the solvers' hyperparameters and compared these "tuned" solvers on an extensive benchmark test set comprising over 600 problem instances and unifying 8 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 performance of the algorithm is directly negatively influenced. Therefore, it is recommended to try different subsolvers, as some subsolvers have strengths and weaknesses depending on the problem class 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 amount of numerical difficulties in the solution of the subproblems. This is partially caused by the fact that with each level, stronger tolerances must be enforced. Additionally, in later iterations, the lower-level problem must be solved more and more accurately, as it becomes more likely to be inconclusive as the computed lower and upper bounds are bracketing zero. This is due to the fact that in later iterations, the computed solution becomes less and less infeasible, e.g., for SIPs $\varepsilon^a \to 0$.

As a result, the user should always critically challenge their decision of the chosen optimality tolerances as they might be overly restrictive compared to their model accuracy because restrictive optimality tolerances greatly influence the solution time. Additionally, they should explore if a solution approach, which guarantees a feasible point upon termination, is necessary or if an ε^a-feasible point (with $\varepsilon^a > 0$) 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 hard 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 benchmark library consists of a large portion of easy problems, some hard problems, and only a small number of medium hard problems (for the tested solvers). It seems that many of the problems constructed as a challenge in previous publications are fast and easy 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 to formulate the suproblems independently in an opinionated way.

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

Apart from further expanding the benchmark test set, we believe there are three main promising directions for improvement, i.e., initialization strategies of the discretization, subproblem formulations and adaptions, and algorithm adaptions, includ-

ing heuristics and extensions. One could apply an (advanced) initialization of the discretized sets instead of starting, e.g., 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 further adapted through using better suitable formulations of the subproblems for the used subsolver, adding KKT conditions in the upper-level problems or through considering higher order-cuts, c.f., [25]. Note that the first proposed adaption 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 implemented, e.g., the interval-based method of [12, 13]. For SIPs, promising ideas for algorithm adaptions include employing the solver *B&F* for SIPs, and then after a given feasibility tolerance is met, use (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 numbers of solved problems. Alternatively, one could simply solve (LLP) with (GSIP-REF) every n iterations and populate the discretization with its solution. The impact of all these adaptions can be easily evaluated using libDIPS and the presented benchmark test set.

Appendices

A Further Subproblem Formulations

A.1 SIP solver

(ORA) of *Oracle* can be reformulated as

$$\begin{aligned} \min_{\boldsymbol{x}\in\mathcal{X},\nu} \quad & \nu \\ \text{s.t.} \quad & f(\boldsymbol{x}) - f^t \leqslant \nu \\ & \boldsymbol{g}^u\left(\boldsymbol{x}, \boldsymbol{y}^k\right) \leqslant \nu \cdot \mathbf{1},\ \forall\ \boldsymbol{y}^k \in \mathcal{Y}^{ORA} \\ & \mathcal{X} \coloneqq \left\{\boldsymbol{x} \in \left[\boldsymbol{x}^{lb}, \boldsymbol{x}^{ub}\right] \subsetneq \mathbb{R}^{n_x} : \boldsymbol{v}^{iu}(\boldsymbol{x}) \leqslant \mathbf{0},\ \boldsymbol{v}^{eu}(\boldsymbol{x}) = \mathbf{0}\right\}. \end{aligned} \tag{ORA-REF}$$

A.2 ESIP solver

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

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

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

$$\begin{aligned} \max_{\boldsymbol{y}\in\mathcal{Y}} \min_{\boldsymbol{z}\in\mathcal{Z}^{MLP}} \quad & \max_{j\in\left\{1\ldots n_{gu}\right\}} g_j^u(\bar{\boldsymbol{x}}, \boldsymbol{y}, \boldsymbol{z}) \\ \text{s.t.} \quad & \mathcal{Y} \coloneqq \left\{\boldsymbol{y} \in \left[\boldsymbol{y}^{lb}, \boldsymbol{y}^{ub}\right] \subsetneq \mathbb{R}^{n_y} : \boldsymbol{v}^{il}(\boldsymbol{y}) \leqslant \mathbf{0},\ \boldsymbol{v}^{el}(\boldsymbol{y}) = \mathbf{0}\right\}. \end{aligned} \tag{ESIP-MLP}$$

A.3 BLP solvers

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

$$\begin{aligned} \min_{\boldsymbol{y}\in\mathcal{Y}(\S)} \quad & h(\boldsymbol{x}, \boldsymbol{y}) \\ \text{s.t.} \quad & \mathcal{Y}(\boldsymbol{x}) \coloneqq \left\{\boldsymbol{y} \in \left[\boldsymbol{y}^{lb}, \boldsymbol{y}^{ub}\right] \subsetneq \mathbb{R}^{n_y} : \boldsymbol{g}^l(\boldsymbol{x}, \boldsymbol{y}) \leqslant \mathbf{0},\ \boldsymbol{v}^{il}(\boldsymbol{y}) \leqslant \mathbf{0},\ \boldsymbol{v}^{el}(\boldsymbol{y}) = \mathbf{0}\right\}. \end{aligned} \tag{BLP-LLP}$$

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

$$\begin{aligned} \min_{\boldsymbol{y}\in\mathcal{Y}(\bar{\boldsymbol{x}}),u\in\mathbb{R}} \quad & u \\ & h(\bar{\boldsymbol{x}}, \boldsymbol{y}) \leqslant \bar{h} + \varepsilon^{AUX} \\ & \boldsymbol{g}^l(\bar{\boldsymbol{x}}, \boldsymbol{y}) \leqslant u \cdot \mathbf{1} \\ & \mathcal{Y}(\boldsymbol{x}) \coloneqq \left\{\boldsymbol{y} \in \left[\boldsymbol{y}^{lb}, \boldsymbol{y}^{ub}\right] \subsetneq \mathbb{R}^{n_y} : \boldsymbol{g}^l(\boldsymbol{x}, \boldsymbol{y}) \leqslant \mathbf{0},\ \boldsymbol{v}^{il}(\boldsymbol{y}) \leqslant \mathbf{0},\ \boldsymbol{v}^{el}(\boldsymbol{y}) = \mathbf{0}\right\} \end{aligned} \tag{BLP-AUX}$$

The auxiliary problem for *BLP-Box* for deciding whether or not a given box $\mathcal{X}_{box}$ 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

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

Note that in [79] uses interval analysis to determine if $\mathcal{X}_{box}$ is valid.

B Definitions

B.1 SIP-Slater Point

Definition 1 (*SIP-Slater Point*) A point $\boldsymbol{x}^S \in \mathcal{X}$ is called an SIP-Slater point in (SIP) if

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

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

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

c.f., Definition A.1 in [77].

B.2 ε^f-optimal SIP-Slater Point

Definition 2 A point $\boldsymbol{x}^S \in \mathcal{X}$ is called an ε^f-optimal SIP-Slater point in (SIP) if

$$f\left(\boldsymbol{x}^S\right) \leqslant f^* + \varepsilon^f \wedge \boldsymbol{g}^u\left(\boldsymbol{x}^S, \boldsymbol{y}\right) \leqslant -\varepsilon^S,\ \forall\ \boldsymbol{y} \in \mathcal{Y}$$

with $\varepsilon^f, \varepsilon^S > 0$, c.f., Lemma 2.4 in [77].

B.3 GSIP-LLP-Slater Point

Definition 3 (*GSIP-LLP-Slater Point*) A point $\boldsymbol{y}^S \in \mathcal{Y}$ is called a GSIP-LLP-Slater point at $\bar{\boldsymbol{x}}$ in (GSIP) if

$$\boldsymbol{g}^l\left(\bar{\boldsymbol{x}}, \boldsymbol{y}^S\right) \leqslant -\varepsilon^S,$$

with some $\varepsilon^S > 0$, c.f., Assumption 4 in [78].

B.4 ε^a-(G)SIP-Feasible Point

Definition 4 A point $\bar{\boldsymbol{x}} \in \mathcal{X}$ is ε^a-SIP-feasible in (SIP) if

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

with $\varepsilon^a \geqslant 0$.

Definition 5 A point $\bar{\boldsymbol{x}} \in \mathcal{X}$ is ε^a-GSIP-feasible in (GSIP) if

$$\min\left\{g_i^u(\boldsymbol{x}, \boldsymbol{y}), \min_{j \in \{1 \dots n_{gl}\}} - g_j^l(\boldsymbol{x}, \boldsymbol{y})\right\} \leqslant \varepsilon^a,\ \forall\ i \in \{1 \dots n_{gu}\},\ \forall\ \boldsymbol{y} \in \mathcal{Y}$$

with $\varepsilon^a \geqslant 0$.

Definition 6 A point $\bar{\boldsymbol{x}} \in \mathcal{X}$ is called (G)SIP-feasible in (SIP) ((GSIP)) if it fulfills Definition 4 (Definition 5) with $\varepsilon^a = 0$.

Acknowledgements 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".
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.
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.

References

[1] E. J. Anderson and A. S. Lewis. An extension of the simplex algorithm for semi-infinite linear programming. *Math. Program.*, 44(1-3):247–269, 1989. ISSN 0025-5610. doi: 10.1007/BF01587092.

[2] D. O. Andreassen and G. A. Watson. Linear Chebyshev approximation without Chebyshev sets. *Bit*, 16(4):349–362, 1976. ISSN 0006-3835. doi: 10.1007/BF01932717.

[3] A. C. Atkinson and V. V. Federov. The design of experiments for discriminating between two rival models. *Biometrika*, 62(1):57–70, 1975. ISSN 0006-3444. doi: 10.1093/biomet/62.1.57.

[4] A. Auslender, A. Ferrer, M. A. Goberna, and M. A. López. Comparative study of RPSALG algorithm for convex semi-infinite programming. *Comput. Optim. Appl.*, 60(1):59–87, June 2015. ISSN 0926-6003. doi: 10.1007/s10589-014-9667-7.

[5] S. Avraamidou and E. N. Pistikopoulos. B-POP: Bi-level parametric optimization toolbox. *Computers & Chemical Engineering*, 122:193–202, 2019. ISSN 0098-1354. doi: 10.1016/j.compchemeng.2018.07.007.

[6] A. Barragán and J.-F. Camacho-Vallejo. An exact algorithm based on the Kuhn–Tucker conditions for solving linear generalized semi-infinite programming problems. *J. Math.*, 2022:1–14, 2022. ISSN 2314-4629. doi: 10.1155/2022/1765385.

[7] I. Barrodale, L. M. Delves, and J. C. Mason. Linear Chebyshev approximation of complex-valued functions. *Math. Comput.*, 32(143):853, 1978. ISSN 0025-5718. doi: 10.2307/2006490.

[8] A. Ben-Tal and A. Nemirovski. Robust solutions of uncertain linear programs. *Oper. Res. Lett.*, 25(1):1–13, 1999. ISSN 0167-6377. doi: 10.1016/S0167-6377(99)00016-4.

[9] A. Ben-Tal and A. Nemirovski. Robust optimization – methodology and applications. *Math. Program.*, 92(3):453–480, 2002. ISSN 0025-5610. doi: 10.1007/s101070100286.

[10] B. Betro. An accelerated central cutting plane algorithm for linear semi-infinite programming. *Math. Program.*, 101(3):479–495, 2004. ISSN 0025-5610. doi: 10.1007/s10107-003-0492-5.

[11] B. Beykal, S. Avraamidou, I. P. E. Pistikopoulos, M. Onel, and E. N. Pistikopoulos. DOMINO: Data-driven Optimization of bi-level Mixed-Integer NOnlinear Problems. *J. Global Optim.*, 78(1):1–36, 2020. ISSN 0925-5001. doi: 10.1007/s10898-020-00890-3.

[12] B. Bhattacharjee, W. H. Green Jr., and P. I. Barton. Interval methods for semi-infinite programs. *Comput. Optim. Appl.*, 30(1):63–93, 2005. ISSN 0926-6003. doi: 10.1007/s10589-005-4556-8.

[13] B. Bhattacharjee, P. Lemonidis, W. H. Green Jr., and P. I. Barton. Global solution of semi-infinite programs. *Math. Program.*, 103(2):283–307, 2005. ISSN 0025-5610. doi: 10.1007/s10107-005-0583-6.

[14] J. W. Blankenship and J. E. Falk. Infinitely constrained optimization problems. *J. Optimiz. Theory App.*, 19(2):261–281, 1976. ISSN 0022-3239. doi: 10.1007/BF00934096.

[15] D. Bongartz, J. Najman, S. Sass, and A. Mitsos. MAiNGO – McCormick-based Algorithm for mixed-integer Nonlinear Global Optimization. Technical report, Process Systems Engineering (AVT.SVT), RWTH Aachen University, 2018. URL http://www.avt.rwth-aachen.de/global/show_document.asp?id=aaaaaaaaabclahw. accessed 07/20/2023.

[16] J. F. Bonnans and A. Shapiro. *Perturbation analysis of optimization problems*. Springer series in operations research. Springer, New York, 2000. ISBN 0387987053. doi: 10.1007/978-1-4612-1394-9.

[17] M. J. Cánovas, M. A. López, J. Parra, and M. I. Todorov. Stability and well-posedness in linear semi-infinite programming. *SIAM Journal on Optimization*, 10(1):82–98, jan 1999. ISSN 1052-6234. doi: 10.1137/S1052623497319869.

[18] M. Cerulli, A. Oustry, C. D'Ambrosio, and L. Liberti. Convergent algorithms for a class of convex semi-infinite programs. *SIAM Journal on Optimization*, 32 (4):2493–2526, oct 2022. ISSN 1052-6234. doi: 10.1137/21M1431047.

[19] Z. Chen. *Optimality Conditions of Semi-Infinite Programming and Generalized Semi-Infinite Programming*. Ph.d. thesis, The Hong Kong Polytechnic University, Hong Kong, 2013.

[20] P. A. Clark and A. W. Westerberg. Bilevel programming for steady-state chemical process design – I. fundamentals and algorithms. *Computers & Chemical Engineering*, 14(1):87–97, 1990. ISSN 0098-1354. doi: 10.1016/0098-1354(90)87007-C.

[21] I. D. Coope and C. J. Price. Exact penalty function methods for nonlinear semi-infinite programming. In P. Pardalos, R. Horst, R. Reemtsen, and J.-J. Rückmann, editors, *Semi-Infinite Programming*, volume 25 of *Nonconvex Optimization and Its Applications*, pages 137–157. Springer US, Boston, MA, 1998. ISBN 978-1-4419-4795-6. doi: 10.1007/978-1-4757-2868-2_5.

[22] I. D. Coope and G. A. Watson. A projected Lagrangian algorithm for semi-infinite programming. *Math. Program.*, 32(3):337–356, 1985. ISSN 0025-5610. doi: 10.1007/BF01582053.

[23] A. R. Curtis and M. J. D. Powell. Necessary conditions for a minimax approximation. *Comput. J.*, 8(4):358–361, 1966. ISSN 0010-4620. doi: 10.1093/comjnl/8.4.358.

[24] M. Diehl, B. Houska, O. Stein, and P. Steuermann. A lifting method for generalized semi-infinite programs based on lower level Wolfe duality. *Comput. Optim. Appl.*, 54(1):189–210, 2013. ISSN 0926-6003. doi: 10.1007/s10589-012-9489-4.

[25] H. Djelassi. *Discretization-based algorithms for the global solution of hierarchical programs*. Dissertation, RWTH Aachen University, 2020.

[26] H. Djelassi and A. Mitsos. A hybrid discretization algorithm with guaranteed feasibility for the global solution of semi-infinite programs. *J. Global Optim.*, 68 (2):227–253, 2017. ISSN 0925-5001. doi: 10.1007/s10898-016-0476-7.

[27] H. Djelassi and A. Mitsos. Global solution of semi-infinite programs with existence constraints. *J. Optimiz. Theory App.*, 188(3):863–881, Feb. 2021. ISSN 0022-3239. doi: 10.1007/s10957-021-01813-2.

[28] H. Djelassi, M. Glass, and A. Mitsos. Discretization-based algorithms for generalized semi-infinite and bilevel programs with coupling equality constraints. *J. Global Optim.*, 75(2):341–392, 2019. ISSN 0925-5001. doi: 10.1007/s10898-019-00764-3.

[29] H. Djelassi, A. Mitsos, and O. Stein. Recent advances in nonconvex semi-infinite programming: Applications and algorithms. *EURO Journal on Computational Optimization*, 9:100006, 2021. doi: 10.1016/j.ejco.2021.100006.

[30] B. P. M. Duarte, W. K. Wong, and A. C. Atkinson. A semi-infinite programming based algorithm for determining T–optimum designs for model discrimination. *J. Multivariate Anal.*, 135:11–24, 2015. ISSN 0047-259X. doi: 10.1016/j.jmva.2014.11.006.

[31] Fair Isaac Corporation. FICO Xpress-Optimizer, reference manual, 2023. URL https://www.fico.com/fico-xpress-optimization/docs/. accessed 12/01/2023.

[32] J. E. Falk and K. Hoffman. A nonconvex max-min problem. *Nav. Res. Logist. Q.*, 24(3):441–450, 1977. ISSN 0028-1441. doi: 10.1002/nav.3800240307.

[33] S.-C. Fang, C.-J. Lin, and S.-Y. Wu. On solving convex quadratic semi-infinite programming problems. *Optimization*, 31(2):107–125, 1994. ISSN 0233-1934. doi: 10.1080/02331939408844009.

[34] M. C. Ferris and A. B. Philpott. An interior point algorithm for semi-infinite linear programming. *Math. Program.*, 43(1-3):257–276, 1989. ISSN 0025-5610. doi: 10.1007/BF01582293.

[35] C. A. Floudas and O. Stein. The adaptive convexification algorithm: A feasible point method for semi-infinite programming. *SIAM Journal on Optimization*, 18(4):1187–1208, jan 2008. ISSN 1052-6234. doi: 10.1137/060657741.

[36] GAMS Development Corporation. *General Algebraic Modeling System (GAMS) Release 43.3.0*. Fairfax, VA, USA, 2023. URL http://www.gams.com/. accessed 12/01/2023.

[37] K. Glashoff and S.-Å. Gustafson. *Linear optimization and approximation: An introduction to the theoretical analysis and numerical treatment of semi-infinite programs*, volume 45 of *Applied Mathematical Sciences*. Springer New York, NY, 1 edition, 1983. ISBN 978-1-4612-1142-6. doi: 10.1007/978-1-4612-1142-6.

[38] M. A. Goberna and M. A. López. *Linear semi-infinite optimization*. Wiley series in mathematical methods in practice. Wiley, Chichester, 1998. ISBN 978-0471970408.

[39] S. Görner. *Ein Hybridverfahren zur Lösung nichtlinearer semi-infiniter Optimierungsprobleme*. Dissertation, Technische Universität Berlin, Berlin, 1997.

[40] I. E. Grossmann and R. W. H. Sargent. Optimum design of chemical plants with uncertain parameters. *AIChE Journal*, 24(6):1021–1028, nov 1978. ISSN 0001-1541. doi: 10.1002/aic.690240612.

[41] F. Guerra Vázquez, J.-J. Rückmann, O. Stein, and G. Still. Generalized semi-infinite programming: A tutorial. *J. Comput. Appl. Math.*, 217(2):394–419, 2008. ISSN 0377-0427. doi: 10.1016/j.cam.2007.02.012.

[42] Z. H. Gümüş. Reactive distillation column design with vapor/liquid/liquid equilibria. *Computers & Chemical Engineering*, 21(1-2):983–988, 1997. ISSN 0098-1354. doi: 10.1016/S0098-1354(97)00177-4.

[43] H. Günzel, H. T. Jongen, and O. Stein. On the closure of the feasible set in generalized semi-infinite programming. *Cent. Europ. J. Oper. Re.*, 15(3):271–280, 2007. ISSN 1435-246X. doi: 10.1007/s10100-007-0030-2.

[44] F. Guo and X. Sun. Semidefinite programming relaxations for linear semi-infinite polynomial programming. *Pac. J. Optim.*, 16(3):395–418, 2020. ISSN 1348-9151.

[45] F. Guo and M. Zhang. An SDP method for fractional semi-infinite programming problems with SOS-convex polynomials. *Optim. Lett.*, 2023. ISSN 1862-4472. doi: 10.1007/s11590-023-01974-1.

[46] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023. URL https://www.gurobi.com. accessed 12/01/2023.

[47] S.-Å. Gustafson. On numerical analysis in semi-infinite programming. In A. V. Balakrishnan, M. Thoma, and R. Hettich, editors, *Semi-Infinite Programming*, volume 15 of *Lecture Notes in Control and Information Sciences*, pages 51–65. Springer Berlin Heidelberg, 1979. ISBN 3-540-09479-2. doi: 10.1007/BFb0003883.

[48] S.-Å. Gustafson and K. O. Kortanek. Numerical treatment of a class of semi-infinite programming problems. *Nav. Res. Logist. Q.*, 20(3):477–504, 1973. ISSN 0028-1441. doi: 10.1002/nav.3800200310.

[49] S. M. Harwood. A note on generalized semi-infinite program bounding methods, 2019.

[50] S. M. Harwood and P. I. Barton. How to solve a design centering problem. *Math. Method. Oper. Res.*, 86(1):215–254, 2017. ISSN 1432-2994. doi: 10.1007/s00186-017-0591-3.

[51] R. Hettich. Chebyshev approximation by H-polynomials: A numerical method. *J. Approx. Theory*, 17(1):97–106, 1976. ISSN 0021-9045. doi: 10.1016/0021-9045(76)90114-3.

[52] R. Hettich. A comparison of some numerical methods for semi-infinite programming. In A. V. Balakrishnan, M. Thoma, and R. Hettich, editors, *Semi-Infinite Programming*, volume 15 of *Lecture Notes in Control and Information Sciences*, pages 112–125. Springer Berlin Heidelberg, 1979. ISBN 3-540-09479-2. doi: 10.1007/BFb0003887.

[53] R. Hettich. An implementation of a discretization method for semi-infinite programming. *Math. Program.*, 34(3):354–361, 1986. ISSN 0025-5610. doi: 10.1007/BF01582235.

[54] R. Hettich and W. van Honstede. On quadratically convergent methods for semi-infinite programming. In A. V. Balakrishnan, M. Thoma, and R. Hettich, editors, *Semi-Infinite Programming*, volume 15 of *Lecture Notes in Control and Information Sciences*, pages 97–111. Springer Berlin Heidelberg, 1979. ISBN 3-540-09479-2. doi: 10.1007/BFb0003886.

[55] R. Hettich and P. Zencke. *Numerische Methoden der Approximation und semi-infiniten Optimierung*. Teubner Studienbücher: Mathematik. Teubner and Vieweg+Teubner Verlag, Stuttgart, 1982. ISBN 978-3-519-02063-9. doi: 10.1007/978-3-322-93108-5.

[56] International Business Machines Corporation. *IBM ILOG CPLEX v22.1.1*. Armonk, NY, USA, 2022.

[57] H. T. Jongen, J.-J. Rückmann, and O. Stein. Generalized semi-infinite optimization: A first order optimality condition and examples. *Math. Program.*, 83 (1-3):145–158, 1998. ISSN 0025-5610. doi: 10.1007/BF02680555.

[58] D. Jungen, H. Djelassi, and A. Mitsos. Adaptive discretization-based algorithms for semi-infinite programs with unbounded variables. *Math. Method. Oper. Res.*, 96(1):83–112, 2022. ISSN 1432-2994. doi: 10.1007/s00186-022-00792-y.

[59] N. Kanzi. Lagrange multiplier rules for non-differentiable DC generalized semi-infinite programming problems. *J. Global Optim.*, 56(2):417–430, 2013. ISSN 0925-5001. doi: 10.1007/s10898-011-9828-5.

[60] P.-M. Kleniati and C. S. Adjiman. 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. ISSN 0925-5001. doi: 10.1007/s10898-013-0120-8.

[61] P.-M. Kleniati and C. S. Adjiman. A generalization of the branch-and-sandwich algorithm: From continuous to mixed-integer nonlinear bilevel problems. *Computers & Chemical Engineering*, 72:373–386, 2015. ISSN 0098-1354. doi: 10.1016/j.compchemeng.2014.06.004.

[62] O. I. Kostyukova and T. V. Tchemisova. Sufficient optimality conditions for convex semi-infinite programming. *Optimization Methods & Software*, 25(2): 279–297, 2010. ISSN 1055-6788. doi: 10.1080/10556780902992803.

[63] K.-H. Küfer, O. Stein, and A. Winterfeld. Semi-infinite optimization meets industry: A deterministic approach to gemstone cutting. *Siam News*, 41 (8), 2008.

[64] P. Lemonidis. *Global Optimization Algorithms for Semi-Infinite and Generalized Semi-Infinite Programs*. Ph.d. thesis, Massachusetts Institute of Technology, USA, 2008.

[65] T. Leon and E. Vercher. A purification algorithm for semi-infinite programming. *Eur. J. Oper. Res.*, 57(3):412–420, 1992. ISSN 0377-2217. doi: 10.1016/0377-2217(92)90353-B.

[66] D.-H. Li, L. Qi, J. Tam, and S.-Y. Wu. A smoothing newton method for semi-infinite programming. *J. Global Optim.*, 30(2-3):169–194, 2004. ISSN 0925-5001. doi: 10.1007/s10898-004-8266-z.

[67] S. J. Li. *Semi-infinite programming and semi-definite optimization problems*. Ph.d. thesis, Hong Kong Polytechnic University, 2003.

[68] C.-J. Lin, S.-C. Fang, and S.-Y. Wu. A dual affine scaling based algorithm for solving linear semi-infinite programming problems. In P. Pardalos, R. Horst, D.-Z. Du, and J. Sun, editors, *Advances in Optimization and Approximation*, volume 1 of *Nonconvex Optimization and Its Applications*, pages 217–234. Springer US, Boston, MA, 1994. ISBN 978-1-4613-3631-0. doi: 10.1007/978-1-4613-3629-7_11.

[69] W. Liu and C. Wang. A smoothing Levenberg–Marquardt method for generalized semi-infinite programming. *Comput. Appl. Math.*, 32(1):89–105, mar 2013. ISSN 2238-3603. doi: 10.1007/s40314-013-0013-y.

[70] Y. Liu, K. L. Teo, and S. Ito. A dual parameterization approach to linear-quadratic semi-infinite programming problems. *Optimization Methods & Software*, 10(3):471–495, 1999. ISSN 1055-6788. doi: 10.1080/10556789908805725.

[71] C. G. Lo Bianco and A. Piazzi. A hybrid algorithm for infinitely constrained optimization. *Int. J. Syst. Sci.*, 32(1):91–102, 2001. ISSN 0020-7721. doi: 10.1080/00207720121051.

[72] M. López and G. Still. Semi-infinite programming. *Eur. J. Oper. Res.*, 180(2):491–518, 2007. ISSN 0377-2217. doi: 10.1016/j.ejor.2006.08.045.

[73] A. Marendet, A. Goldsztejn, G. Chabert, and C. Jermann. A standard branch-and-bound approach for nonlinear semi-infinite problems. *Eur. J. Oper. Res.*, 282(2):438–452, 2020. ISSN 0377-2217. doi: 10.1016/j.ejor.2019.10.025.

[74] S. Mehrotra and D. Papp. A cutting surface algorithm for semi-infinite convex programming with an application to moment robust optimization. *SIAM Journal on Optimization*, 24(4):1670–1697, jan 2014. ISSN 1052-6234. doi: 10.1137/130925013.

[75] R. Misener and C. A. Floudas. ANTIGONE: Algorithms for coNTinuous / Integer Global Optimization of Nonlinear Equations. *J. Global Optim.*, 59(2):503–526, Mar. 2014. ISSN 0925-5001. doi: 10.1007/s10898-014-0166-2.

[76] A. Mitsos. Test set of semi-infinite programs. Technical report, Process Systems Engineering (AVT.SVT), RWTH Aachen University, Aachen, Germany, 2009. URL https://www.avt.rwth-aachen.de/cms/AVT/Forschung/Systemverfahrenstechnik/~kpdo/A-Test-Set-of-Semi-Infinite-Programs/?lidx=1. accessed 08/17/2023.

[77] A. Mitsos. Global optimization of semi-infinite programs via restriction of the right-hand side. *Optimization*, 60(10-11):1291–1308, 2011. ISSN 0233-1934. doi: 10.1080/02331934.2010.527970.

[78] A. Mitsos and A. Tsoukalas. Global optimization of generalized semi-infinite programs via restriction of the right hand side. *J. Global Optim.*, 61(1):1–17, 2015. ISSN 0925-5001. doi: 10.1007/s10898-014-0146-6.

[79] A. Mitsos, P. Lemonidis, and P. I. Barton. Global solution of bilevel programs with a nonconvex inner program. *J. Global Optim.*, 42(4):475–513, dec 2008. ISSN 0925-5001. doi: 10.1007/s10898-007-9260-z.

[80] A. Mitsos, P. Lemonidis, C. K. Lee, and P. I. Barton. Relaxation-based bounds for semi-infinite programs. *SIAM Journal on Optimization*, 19(1):77–113, jan 2008. ISSN 1052-6234. doi: 10.1137/060674685.

[81] L.-P. Pang and Q. Wu. A feasible proximal bundle algorithm with convexification for nonsmooth, nonconvex semi-infinite programming. *Numer. Algorithms*, 90 (1):387–422, 2022. ISSN 1017-1398. doi: 10.1007/s11075-021-01192-9.

[82] L.-P. Pang, J. Lv, and J.-H. Wang. Constrained incremental bundle method with partial inexact oracle for nonsmooth convex semi-infinite programming problems. *Comput. Optim. Appl.*, 64(2):433–465, 2016. ISSN 0926-6003. doi: 10.1007/s10589-015-9810-0.

[83] P. Pardalos, R. Horst, R. Reemtsen, and J.-J. Rückmann, editors. *Semi-Infinite Programming*, volume 25. Springer US, Boston, MA, 1998. doi: 10.1007/978-1-4757-2868-2.

[84] R. Paulavičius and C. S. Adjiman. BASBLib – a library of bilevel test problems. *Computers & Chemical Engineering*, 132:106609, 2020. ISSN 0098-1354. doi: 10.5281/zenodo.3266835.

[85] E. Polak, L. Qi, and D. Sun. First-order algorithms for generalized semi-infinite min-max problems. *Comput. Optim. Appl.*, 13(1/3):137–161, 1999. ISSN 0926-6003. doi: 10.1023/A:1008660924636.

[86] M. J. D. Powell. Karmarkar's algorithm: a view from nonlinear programming, 1989. URL https://ir.canterbury.ac.nz/items/50cd806f-63c4-456b-a66e-132f78653b64.

[87] C. J. Price. *Non-linear semi-infinite programming.* Ph.d. thesis, University of Canterbury, New Zealand, 1992.

[88] C. J. Price and I. D. Coope. Numerical experiments in semi-infinite programming. *Comput. Optim. Appl.*, 6(2):169–189, 1996. ISSN 0926-6003. doi: 10.1007/BF00249645.

[89] L. Qi, S.-Y. Wu, and G. Zhou. Semismooth newton methods for solving semi-infinite programming problems. *J. Global Optim.*, 27(2/3):215–232, 2003. ISSN 0925-5001. doi: 10.1023/A:1024814401713.

[90] E. I. Remez. *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, Tenn., 1962.

[91] J. O. Royset, E. Polak, and A. Kiureghian. Adaptive approximations and exact penalization for the solution of generalized semi-infinite min-max problems. *SIAM Journal on Optimization*, 14(1):1–34, jan 2003. ISSN 1052-6234. doi: 10.1137/S1052623402406777.

[92] J.-J. Rückmann and A. Shapiro. First-order optimality conditions in generalized semi-infinite programming. *J. Optimiz. Theory App.*, 101(3):677–691, 1999. ISSN 0022-3239. doi: 10.1023/A:1021746305759.

[93] J.-J. Rückmann and A. Shapiro. Second-order optimality conditions in generalized semi-infinite programming. *Set-Valued Var. Anal.*, 9(1/2):169–186, 2001. ISSN 1877-0533. doi: 10.1023/A:1011239607220.

[94] J.-J. Rückmann and A. Shapiro. Augmented Lagrangians in semi-infinite programming. *Math. Program.*, 116(1-2):499–512, 2009. ISSN 0025-5610. doi: 10.1007/s10107-007-0115-7.

[95] E. Rudnick-Cohen, J. W. Herrmann, and S. Azarm. Non-convex feasibility robust optimization via scenario generation and local refinement. *J. Mech. Design*, 142(5), 2020. ISSN 1050-0472. doi: 10.1115/1.4044918.

[96] N. V. Sahinidis. BARON 2023.6.23: Global Optimization of Mixed-Integer Nonlinear Programs, *User's Manual*, 2023. URL http://www.minlp.com/downloads/docs/baron%20manual.pdf. accessed 04/09/2023.

[97] J. Schwientek, T. Seidel, and K.-H. Küfer. A transformation-based discretization method for solving general semi-infinite optimization problems. *Math. Method. Oper. Res.*, 93(1):83–114, 2021. ISSN 1432-2994. doi: 10.1007/s00186-020-00724-8.

[98] T. Seidel and K.-H. Küfer. An adaptive discretization method solving semi-infinite optimization problems with quadratic rate of convergence. *Optimization*, 71(8):2211–2239, 2019. ISSN 0233-1934. doi: 10.1080/02331934.2020.1804566.

[99] T. Seidel and J. Schwientek. GSIPLib&Gen: A library and generator of general semi-infinite programming test problems: Version 1.0, 2017. URL https://www.itwm.fraunhofer.de/en/departments/optimization/products-and-services/gsip-lib-and-gen.html.

[100] A. G. W. Selassie. *A coarse solution of generalized semi-infinite optimization problems via robust analysis of marginal functions and global optimization.* Dissertation, Technischen Universitat Ilmenau, 2005. URL https://www.db-thueringen.de/receive/dbt_mods_00002883.

[101] A. Sinha, P. Malo, and K. Deb. A review on bilevel optimization: From classical to evolutionary approaches and applications. *IEEE Transactions on Evolutionary Computation*, 22(2):276–295, apr 2018. ISSN 1089-778X. doi: 10.1109/TEVC.2017.2712906.

[102] J. Spjøtvold, P. Tøndel, and T. A. Johansen. A method for obtaing continuos solutions to multiparametric linear programs. *IFAC Proceedings Volumes*, 38(1):253–258, 2005. ISSN 1474-6670. doi: 10.3182/20050703-6-CZ-1902.00903.

[103] O. Stein and P. Pardalos. *Bi-level strategies in semi-infinite programming*, volume 71 of *Nonconvex Optimization and Its Applications*. Springer Science+Business Media, Boston, MA, 2003. ISBN 978-1-4613-4817-7. doi: 10.1007/978-1-4419-9164-5.

[104] O. Stein and P. Steuermann. The adaptive convexification algorithm for semi-infinite programming with arbitrary index sets. *Math. Program.*, 136(1):183–207, 2012. ISSN 0025-5610. doi: 10.1007/s10107-012-0556-5.

[105] O. Stein and G. Still. On generalized semi-infinite optimization and bilevel optimization. *Eur. J. Oper. Res.*, 142(3):444–462, 2002. ISSN 0377-2217. doi: 10.1016/S0377-2217(01)00307-1.

[106] G. Still. Generalized semi-infinite programming: Theory and methods. *Eur. J. Oper. Res.*, 119(2):301–313, 1999. ISSN 0377-2217. doi: 10.1016/S0377-2217(99)00132-0.

[107] G. Still. Generalized semi-infinite programming: numerical aspects. *Optimization*, 49(3):223–242, 2001. ISSN 0233-1934. doi: 10.1080/02331930108844531.

[108] G. Still. Discretization in semi-infinite programming: the rate of convergence. *Math. Program.*, 91(1):53–69, 2001. ISSN 0025-5610. doi: 10.1007/s101070100239.

[109] G. Still. Optimization problems with infinitely many constraints. *Carpathian J. Math.*, 18(2):343–354, 2002. ISSN 1222-1201.

[110] R. L. Streit and A. H. Nuttall. A general Chebyshev complex function approximation procedure and an application to beamforming. *J. Acoust. Soc. Am.*, 72 (1):181–190, 1982. ISSN 0001-4966. doi: 10.1121/1.388002.

[111] K. Su, C. Xu, and L. Ren. Filter trust region method for nonlinear semi-infinite programming problem. *Math. Probl. Eng.*, 2018:1–9, 2018. ISSN 1024-123X. doi: 10.1155/2018/3921592.

[112] R. E. Swaney and I. E. Grossmann. An index for operational flexibility in chemical process design. part i: Formulation and theory. *Aiche J.*, 31(4):621–630, 1985. ISSN 0001-1541. doi: 10.1002/aic.690310412.

[113] Y. Tanaka, M. Fukushima, and T. Ibaraki. A globally convergent SQP method for semi-infinite nonlinear optimization. *J. Comput. Appl. Math.*, 23(2):141–153, 1988. ISSN 0377-0427. doi: 10.1016/0377-0427(88)90276-2.

[114] K. L. Teo, X. Q. Yang, and L. S. Jennings. Computational discretization algorithms for functional inequality constrained optimization. *Ann. Oper. Res.*, 98 (1/4):215–234, 2000. ISSN 0254-5330. doi: 10.1023/A:1019260508329.

[115] A. Tezel Özturan. Solving generalized semi-infinite programming problems with a trust region method. *Acta Phys. Pol. A*, 128(2B):B–93–B–97, aug 2015. ISSN 0587-4246. doi: 10.12693/APhysPolA.128.B-93.

[116] R. Tichatschke and V. Nebeling. A cutting-plane method for quadratic semi infinite programming problems. *Optimization*, 19(6):803–817, 1988. ISSN 0233-1934. doi: 10.1080/02331938808843393.

[117] A. Tsoukalas and B. Rustem. A feasible point adaptation of the Blankenship and Falk algorithm for semi-infinite programming. *Optim. Lett.*, 5(4):705–716, 2011. ISSN 1862-4472. doi: 10.1007/s11590-010-0236-4.

[118] A. Tsoukalas, B. Rustem, and E. N. Pistikopoulos. 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. ISSN 0925-5001. doi: 10.1007/s10898-008-9321-y.

[119] A. I. F. Vaz, E. Fernandes, and M. Gomes. Nsips: Nonlinear semi-infinite programming solver. Technical report, Technical Report ALG/EF/1-2001, 2001. http://www. eng. uminho. pt/ dps/aivaz, 2004.

[120] A. I. F. Vaz, E. M. G. P. Fernandes, and M. P. S. F. Gomes. Sipampl. *ACM Trans. Math. Softw.*, 30(1):47–61, mar 2004. ISSN 0098-3500. doi: 10.1145/974781.974784.

[121] F. G. Vázquez and J.-J. Rückmann. Extensions of the Kuhn–Tucker constraint qualification to generalized semi-infinite programming. *SIAM Journal on Optimization*, 15(3):926–937, jan 2005. ISSN 1052-6234. doi: 10.1137/S1052623403431500.

[122] A. Wächter and L. T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Math. Program.*, 106(1):25–57, 2006. ISSN 0025-5610. doi: 10.1007/s10107-004-0559-y.

[123] G. A. Watson. A multiple exchange algorithm for multivariate Chebyshev approximation. *SIAM Journal on Numerical Analysis*, 12(1):46–52, mar 1975. ISSN 0036-1429. doi: 10.1137/0712004.

[124] G. A. Watson. Numerical experiments with globally convergent methods for semi-infinite programming problems. In A. V. Fiacco and K. O. Kortanek, editors, *Semi-Infinite Programming and Applications*, Lecture Notes in Economics and Mathematical Systems, pages 193–205. Springer Berlin Heidelberg, Berlin, Heidelberg, 1983. ISBN 978-3-642-46477-5.

[125] W. Wiesemann, A. Tsoukalas, P.-M. Kleniati, and B. Rustem. Pessimistic bilevel optimization. *SIAM Journal on Optimization*, 23(1):353–380, jan 2013. doi: 10.1137/120864015.

[126] M. E. Wilhelm and M. D. Stuber. EAGO.jl: easy advanced global optimization in Julia. *Optimization Methods & Software*, 37(2):425–450, 2022. ISSN 1055-6788. doi: 10.1080/10556788.2020.1786566.

[127] A. Winterfeld. Application of general semi-infinite programming to lapidary cutting problems. *Eur. J. Oper. Res.*, 191(3):838–854, 2008. ISSN 0377-2217. doi: 10.1016/j.ejor.2007.01.057.

[128] S.-Y. Wu, D.-H. Li, L. Qi, and G. Zhou. An iterative method for solving KKT system of the semi-infinite programming. *Optimization Methods & Software*, 20 (6):629–643, 2005. ISSN 1055-6788. doi: 10.1080/10556780500094739.

[129] Y. Xu, W. Sun, and L. Qi. On solving a class of linear semi-infinite programming by SDP method. *Optimization*, pages 1–14, 2013. ISSN 0233-1934. doi: 10.1080/02331934.2013.793325.

[130] S. Žaković and B. Rustem. Semi-infinite programming and applications to minimax problems. *Ann. Oper. Res.*, 124(1-4):81–110, 2003. ISSN 0254-5330. doi: 10.1023/B:ANOR.0000004764.76984.30.

[131] L. Zhang, S.-Y. Wu, and M. A. López. A new exchange method for convex semi-infinite programming. *SIAM Journal on Optimization*, 20(6):2959–2977, jan 2010. ISSN 1052-6234. doi: 10.1137/090767133.

[132] J. L. Zhou and A. L. Tits. An SQP algorithm for finely discretized continuous minimax problems and other minimax problems with many objective functions. *SIAM Journal on Optimization*, 6(2):461–487, may 1996. ISSN 1052-6234. doi: 10.1137/0806025.

[133] S. Zhou, A. B. Zemkoho, and A. Tin. BOLIB: Bilevel Optimization LIBrary of Test Problems. In S. Dempe and A. B. Zemkoho, editors, *Bilevel optimization*, volume 161 of *Springer optimization and its applications*, pages 563–580. Springer, Cham, Switzerland, 2020. ISBN 978-3-030-52119-6. doi: 10.1007/978-3-030-52119-6_19.

[134] A. Zingler, D. Jungen, H. Djelassi, and A. Mitsos. *libALE – a library for algebraic-logical expression trees*, 2022. URL https://git.rwth-aachen.de/avt.svt/public/libale.git. accessed 07/19/2022.