Molan, Ioana; Schmidt, Martin

Using neural networks to solve linear bilevel problems with unknown lower level

Optimization Letters

Provided in Cooperation with:
Springer Nature

Suggested Citation: Molan, Ioana; Schmidt, Martin (2023) : Using neural networks to solve linear bilevel problems with unknown lower level, Optimization Letters, ISSN 1862-4480, Springer, Berlin, Heidelberg, Vol. 17, Iss. 5, pp. 1083-1103, https://doi.org/10.1007/s11590-022-01958-7

Optimization Letters (2023) 17:1083–1103
https://doi.org/10.1007/s11590-022-01958-7

ORIGINAL PAPER

Using neural networks to solve linear bilevel problems with unknown lower level

Ioana Molan[1] · **Martin Schmidt**[1]

Received: 1 July 2022 / Accepted: 6 December 2022 / Published online: 16 February 2023

Abstract
Bilevel problems are used to model the interaction between two decision makers in which the lower-level problem, the so-called follower's problem, appears as a constraint in the upper-level problem of the so-called leader. One issue in many practical situations is that the follower's problem is not explicitly known by the leader. For such bilevel problems with unknown lower-level model we propose the use of neural networks to learn the follower's optimal response for given decisions of the leader based on available historical data of pairs of leader and follower decisions. Integrating the resulting neural network in a single-level reformulation of the bilevel problem leads to a challenging model with a black-box constraint. We exploit Lipschitz optimization techniques from the literature to solve this reformulation and illustrate the applicability of the proposed method with some preliminary case studies using academic and linear bilevel instances.

Keywords Bilevel optimization · Unknown follower problems · Neural networks · Lipschitz optimization

1 Introduction

Bilevel optimization has been a highly active field of research in the last decades and has gained increasing attention over the last years. The main reason is that this class of optimization problems can serve as a powerful modeling tool in situations in which one has to model hierarchical decision making; see, e.g., the recent surveys by Beck et al. [18] and Kleinert et al. [2] as well as the annotated bibliography by Dempe [10] to get an overview of the many applications of bilevel optimization. However, this ability also renders bilevel optimization problems very hard to solve

✉ Martin Schmidt
martin.schmidt@uni-trier.de

Ioana Molan
molan@uni-trier.de

[1] Department of Mathematics, Trier University, Universitätsring 15, 54296 Trier, Germany

both in theory and practice. For instance, even linear bilevel problems are strongly NP-hard [15].

As discussed in the survey by Beck et al. [2], bilevel optimization problems can also be subject to uncertainty. Moreover, the sources of potential uncertainties are even richer compared to usual, i.e., single-level, optimization. The reason is that not only the problem's data can be uncertain but also the (observation of the) decisions of the two players can be noisy or uncertain. For instance, it might be the case that the leader is not sure about whether the follower can solve the respective lower-level problem to global optimality and might want to hedge against possibly occurring near-optimal solutions; see, e.g., Besançon et al. [3, 4]. In this short paper, we go even one step further and assume that the leader has no knowledge about an explicit formulation of the lower-level problem. Hence, the leader needs to solve a bilevel optimization problem and the lower level is unknown. What might sound impossible at a first glance can be done, at least approximately, if the bilevel game is repeatedly played so that past pairs of leader decisions and respective responses of the follower can be observed and collected. The obtained set of pairs of decisions can then be used as training data to train a neural network that learns the best-response function of the follower.

The downside of this approach is that the input-output mapping of the neural network can again not be stated using a closed-form expression. This leads to the field of optimization under black-box constraints [23]. Fortunately, there is some recent research on deep neural networks with specific activation functions that shows that the input-output mapping of the neural network is Lipschitz continuous and that Lipschitz constants can actually be computed using specifically chosen semidefinite programs [11, 22]. This paves the way for applying the Lipschitz optimization method proposed in Schmidt et al. [25], see Schmidt et al. [26] as well, in order to solve the single-level reformulation of the bilevel problem with unknown lower level in which the optimal response of the follower is replaced by the mapping that describes the input-output behavior of the trained neural network. In the Lipschitz-based approach, the only nonlinearity, which is the one given by the input-output mapping of the neural network, is outer-approximated using the Lipschitz constant of the mapping. This outer approximation is tightened from iteration to iteration, leading to a union of polyhedra that form a relaxation of the graph of the nonlinearity, which enables the application of powerful state-of-the-art mixed-integer solvers.

The main contribution of this short note is the combination of the recent results about the Lipschitz continuity of special neural networks with recent publications on Lipschitz optimization. By carrying this out in a careful way and by slightly adapting the method proposed in Schmidt et al. [25], we obtain a convergent algorithm for this highly challenging situation. We further illustrate the applicability of our approach in a case study based on academic bilevel instances from the literature.

Although there have been some applications of bilevel optimization problems for machine learning (see, e.g., Table 1 in Khanduri et al. [17]), this is, to the best of our knowledge, the first paper that uses neural networks to solve general bilevel optimization problems. The only other paper we are aware of following this idea is the one by Vlah et al. [27], who apply deep convolutional neural networks to classic bilevel bidding problems in power markets. However, they focus on the specific application

and the specific type of network and its training. In contrast, we focus on the general mathematical idea of replacing unknown lower levels with neutral networks. Furthermore, the work by Borrero et al. [7] presents a sequential learning method for linear bilevel problems under incomplete information. Provided that the strategic players interact with each other multiple times, the authors develop feedback mechanisms to update the missing information for the lower-level objective. Nevertheless, this way to address uncertainty and learning clearly differs from our neural-network based approach. The main idea of this short paper is to give a proof of concept for the proposed method and we, thus, make some simplifications such as that we only consider linear bilevel problems with a scalar upper-level variable and without coupling constraints. We discuss these assumptions in more detail in Sect. 2 and how the setting can be generalized in Sect. 6. Let us finally comment on that our approach is related to other methods for solving bilevel optimization problems that rely on using optimal-value or best-response functions (see, e.g., Lozano and Smith [20] and the references therein). However, our approach is different since we do not work with these functions themselves but with surrogates that we obtain from training a neural network.

The remainder of the paper is structured as follows. In Sect. 2, we introduce the class of bilevel problems that we consider and pose the main assumptions that are required in what follows. Afterward, in Sect. 3, we then review the recent literature about computing Lipschitz constants for deep neural networks, which is a prerequisite for the overall solution approach discussed in Sect. 4. Section 5 contains the case studies that illustrate the applicability of our approach for small instances from the literature. Finally, we conclude in Sect. 6 and discuss some topics for future research.

2 Problem statement

We consider linear bilevel problems of the general form

$$\min_{x\in X,y} \quad c^\top x + d^\top y \tag{1a}$$
$$\text{s.t.} \quad Ax \geq a, \tag{1b}$$
$$\underline{x} \leq x \leq \bar{x}, \quad x \in \mathbb{R}^{n_x}, \tag{1c}$$
$$y \in \Psi(x), \tag{1d}$$

where $\Psi(x)$ is the set of optimal solutions of the x-parameterized problem

$$\min_{y\in Y} \quad f^\top y \tag{2a}$$
$$\text{s.t.} \quad Cx + Dy \geq b, \tag{2b}$$

with $c \in \mathbb{R}^{n_x}$, $d, f \in \mathbb{R}^{n_y}$, $A \in \mathbb{R}^{m\times n_x}$, $C \in \mathbb{R}^{\ell\times n_x}$, $D \in \mathbb{R}^{\ell\times n_y}$, $a \in \mathbb{R}^m$, and $b \in \mathbb{R}^\ell$. Problem (1) is called the upper-level or leader's problem. The decision variables of

the leader are $x \in \mathbb{R}^{n_x}$. Problem (2) is called the lower-level or follower's problem and has the decision variables $y \in \mathbb{R}^{n_y}$. Note that we consider the optimistic bilevel problem. Hence, the leader also optimizes over y if the lower-level problem is not uniquely solvable. The set $\Omega := \{(x, y) \in X \times Y : (1b), (1c), (2b)\}$ is called the shared constraint set and the set $\mathcal{F} := \{(x, y) \in \Omega : y \in \Psi(x)\}$ is called the bilevel feasible set. With $\mathcal{F}_x$, we denote its projection onto the x variables. For what follows, we make the ensuing assumptions.

Assumption 1

(i) The upper-level decision $x \in \mathbb{R}^{n_x}$ is scalar, i.e., $n_x = 1$.
(ii) For all upper-level decisions x for which $\Psi(x)$ is non-empty, we have $|\Psi(x)| = 1$. Hence, if the lower level is feasible and bounded, then it has a unique optimal solution y. Consequently, we can write $y = \Psi(x)$ instead of $y \in \Psi(x)$.
(iii) The solution-set mapping $\Psi(\cdot)$ is Lipschitz continuous.

According to Assumption 1, $\Psi(x)$ is a singleton, leading to a one-to-one correspondence between follower's optimal responses and the upper-level decisions. Moreover, the mapping $\Psi(\cdot)$ is polyhedral, i.e., its graph is a finite union of polyhedra; see Theorem 3.1 in Dempe [9]. Furthermore, the bilevel feasible set is connected because we have no coupling constraints and, thus, $\Psi_i(x)$ is a Lipschitz continuous and piecewise linear function in x for all $i \in [n_y] := \{1, \dots, n_y\}$.

Lower-level uniqueness is particularly important when it comes to training a neural network to learn the optimal response of the follower for a given upper-level decision. It ensures that, during supervised learning, there is only a single output y to be learned for a given input x. The assumption of a scalar leader's decision can be generalized and is mainly taken for the ease of presentation. We will discuss this in more detail in our conclusion in Sect. 6.

3 Using neural networks to approximate the follower's response

Stating the bilevel problem (1) relies on the knowledge about an explicit formulation for the lower-level problem (2). In the case in which the follower's problem (2) is not known by the leader but past pairs (x, y) of leader and follower decisions are available, we propose using this historical data to learn the optimal response $y = \Psi(x)$ of the follower using neural networks.

Such (x, y)-pairs naturally arise in many applications. For instance, for pricing models, the upper-level variable x is the price set by the leader for a certain good and y is the amount of this good bought by the follower. Both the price and the bought goods can be observed and collected to obtain (x, y)-pairs to train a neural network. Similar situations appear in many other fields such as in bilevel optimization for market design, transportation, or security. Obviously, it is required that the game modeled by the bilevel problem is played repeatedly so that large enough training sets can be collected over time.

In what follows, we use a neural network to learn $\Psi(\cdot)$, which will then replace the lower level and turn the bilevel model into the single-level problem

$$\begin{aligned} \min_{x,y} \quad & cx + d^\top y && \text{(3a)} \\ \text{s.t.} \quad & Ax \geq a, \quad \underline{x} \leq x \leq \bar{x}, \quad x \in \mathbb{R}, && \text{(3b)} \\ & g_i(x) = y_i, \quad i \in [n_y], && \text{(3c)} \end{aligned}$$

where g_i is the function corresponding to the neural network for the ith follower's response y_i, $i \in [n_y]$. Thus, we assume $g_i(x) \approx \Psi_i(x)$. In what follows, we exploit some recent results from the literature to show that g_i is Lipschitz continuous and that Lipschitz constants can indeed be computed. This property of the used neural networks is vital for the decomposition method we use to solve Problem (3); see Sect. 4.

3.1 Lipschitz constants of neural networks

In this paper, we use the LipSDP-Neuron method published in Fazlyab et al. [11] to compute Lipschitz constants of the neural network functions g_i, $i \in [n_y]$. That is, we compute a constant $L_i \geq 0$ that satisfies

$$|g_i(x_1) - g_i(x_2)| \leq L_i |x_1 - x_2| \quad \text{for all } x_1, x_2 \in \mathbb{R}$$

and for all $i \in [n_y]$. The main idea in Fazlyab et al. [11] is to replace the nonlinear activation functions at the nodes of a neural network by so-called incremental quadratic constraints, which then allows to state the problem of estimating Lipschitz constants as a semidefinite program (SDP). It is worth noting that the most complex and, hence, the most accurate version of LipSDP in Fazlyab et al. [11] is shown to be wrong in Pauli et al. [22]. Thus, we use the second of the three versions of LipSDP, namely LipSDP-Neuron.

We now describe the quadratic constraints that we use to replace all activation functions $\phi(x) = [\varphi(x_1), \ldots, \varphi(x_n)]^\top : \mathbb{R}^n \to \mathbb{R}^n$ in a network layer, where the same slope-restricted function $\varphi : \mathbb{R} \to \mathbb{R}$ is applied to each component of ϕ. Here and in what follows, we call a function φ slope-restricted w.r.t. $0 \leq \alpha < \beta < \infty$ if

$$\alpha \leq \frac{\varphi(y) - \varphi(x)}{y - x} \leq \beta \tag{4}$$

holds for all $x, y \in \mathbb{R}$. This definition states that the slope of the line connecting any two points x and y on the graph of φ is bounded by α and β. It is easy to see that the Rectified Linear Unit (ReLU) activation function defined as $\varphi(x) := \max\{0, x\}$ is slope-restricted with $\alpha = 0$ and $\beta = 1$; see Goodfellow et al. [13]. Furthermore, if the activation function φ is slope-restricted w.r.t. $[\alpha, \beta] = [0, 1]$, then so is the vector-valued function $\phi(x) = [\varphi(x_1), \ldots, \varphi(x_n)]$, which contains all activation functions in a network layer [12], if we apply the definition in (4) component-wise.

The following lemma shows that the slope property (4) can be written as a quadratic constraint. We use the notation $\mathbb{S}^n$ for the set of all symmetric $n \times n$ matrices.

Lemma 1 (Based on Lemma 1 in Fazlyab et al. [11]) *Suppose $\varphi : \mathbb{R} \to \mathbb{R}$ is slope-restricted w.r.t. α and β. Moreover, we define the set*

$$\mathcal{T}_n := \left\{ T \in \mathbb{S}^n \,:\, T = \sum_{i=1}^{n} \lambda_{ii} e_i e_i^\top,\ \lambda_{ii} \geq 0 \right\} \tag{5}$$

of diagonal matrices T with nonnegative entries. Then, for any $T \in \mathcal{T}_n$, the function $\phi(x) = [\varphi(x_1), \dots, \varphi(x_n)]^\top : \mathbb{R}^n \to \mathbb{R}^n$ satisfies the quadratic constraint

$$\begin{pmatrix} x - y \\ \phi(x) - \phi(y) \end{pmatrix}^\top \begin{bmatrix} -2\alpha\beta T & (\alpha+\beta)T \\ (\alpha+\beta)T & -2T \end{bmatrix} \begin{pmatrix} x - y \\ \phi(x) - \phi(y) \end{pmatrix} \geq 0 \tag{6}$$

for all $x, y \in \mathbb{R}^n$.

The statement of the lemma above is based on Lemma 1 in Fazlyab et al. [11] and has been modified according to the corrections published in Pauli et al. [22]. There, the authors give a counterexample that shows that the original lemma in Fazlyab et al. [11] is wrong. To be more specific, they show that there exists a matrix T built according to the definition of the set $\mathcal{T}_n$ given in Fazlyab et al. [11] that violates (6).

Assuming that all activation functions $\varphi : \mathbb{R} \to \mathbb{R}$ are the same, a feed-forward neural network $f(x) : \mathbb{R}^{n_0} \to \mathbb{R}^{\ell+1}$ can be written compactly as

$$B\mathbf{x} = \phi(A\mathbf{x} + b) \quad \text{and} \quad f(x) = C\mathbf{x} + b^\ell, \tag{7}$$

where $\mathbf{x} = [(x^0)^\top, \dots, (x^\ell)^\top]^\top$ is the concatenation of the input values at every layer of the network, ℓ is the number of layers, $x^i \in \mathbb{R}^{n_i}$ for all $i \in \{1, \dots, \ell\}$, ϕ is the vector-valued function, which applies the activation function φ to every entry of the input vector, and

$$A = \begin{bmatrix} W^0 & 0 & \dots & 0 & 0 \\ 0 & W^1 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & W^{\ell-1} & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & I_{n_1} & 0 & \dots & 0 \\ 0 & 0 & I_{n_2} & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & I_{n_\ell} \end{bmatrix},$$
$$C = \begin{bmatrix} 0 & \dots & 0 & W^\ell \end{bmatrix}, \quad b = \left[(b^0)^\top, \dots, (b^{\ell-1})^\top \right]^\top \tag{8}$$

holds, where W^i is the weight matrix connecting layer i with layer $i+1$, and I_{n_i} is the $n_i \times n_i$ identity matrix. The next theorem is the central result in Fazlyab et al. [11]

and shows that the Lipschitz constant of a neural network is the solution of an SDP in which the matrix T defined in Lemma 1 serves as a decision variable.

Theorem 2 (Theorem 2 in Fazlyab et al. [11]) *Consider an ℓ-layer and fully connected neural network given by (7). Let $n = \sum_{k=1}^{\ell} n_k$ be the total number of hidden neurons and suppose that the activation functions are slope-restricted w.r.t. α and β. Define $\mathcal{T}_n$ as in (5), A and B as in (8), and consider the matrix inequality*

$$M(\rho, T) = \begin{bmatrix} A \\ B \end{bmatrix}^{\top} \begin{bmatrix} -2\alpha\beta T & (\alpha+\beta)T \\ (\alpha+\beta)T & -2T \end{bmatrix} \begin{bmatrix} A \\ B \end{bmatrix} + R \preceq 0 \tag{9}$$

with

$$R = \begin{bmatrix} -\rho I_{n_0} & 0 & \dots & 0 & 0 \\ 0 & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & \dots & 0 & (W^{\ell})^{\top} W^{\ell} \end{bmatrix}.$$

If (9) holds for some $(\rho, T) \in \mathbb{R}_{\geq 0} \times \mathcal{T}_n$, then $\|f(x) - f(y)\|_2 \leq \sqrt{\rho}\|x - y\|_2$ holds for all $x, y \in \mathbb{R}^{n_0}$.

As a result of Theorem 2, a Lipschitz constant for multi-layer networks can be computed by solving

$$\min_{\rho, T} \quad \rho \quad \text{s.t.} \quad M(\rho, T) \preceq 0, \quad T \in \mathcal{T}_n, \tag{10}$$

where $(\rho, T) \in \mathbb{R}_+ \times \mathcal{T}_n$ are the decision variables. Furthermore, $M(\rho, T)$ is linear in ρ and T and the set $\mathcal{T}_n$ is convex. Hence, Problem (10) is a semidefinite program.

4 Solution approach

The neural network constraint (3c) turns model (3) into a challenging problem. In particular, for complex and large neural networks, it cannot be expected to get a closed-form expression for g_i, $i \in [n_y]$, that has reasonable properties required for optimization. In any case, the resulting constraints will be nonlinear, nonconvex, and nonsmooth. However, we can evaluate these constraints and we can compute their Lipschitz constants. Thus, it is reasonable to make the following assumption.

Assumption 2 An oracle is available that evaluates $g_i(\cdot)$ for all $i \in [n_y]$ and all g_i are globally Lipschitz continuous on $\underline{x} \leq x \leq \bar{x}$ with known global Lipschitz constant L_i.

To solve Problem (3), we use the decomposition method published in Schmidt et al. [25] but modify it slightly so that we can apply it to our setting. We assume Lipschitz continuity of the problematic nonlinearities and use the corresponding Lipschitz constants to build a mixed-integer linear problem (MILP) that is a relaxation of the original model (3). We refine this relaxation in every iteration until a satisfactory solution is found or until the problem is shown to be infeasible. A satisfactory solution is formally defined to be an ϵ-*feasible* point, i.e., a point that solves

$$\min_{x,y} \quad cx + d^\top y \tag{11a}$$

$$\text{s.t.} \quad Ax \geq a, \quad \underline{x} \leq x \leq \bar{x}, \quad x \in \mathbb{R}, \tag{11b}$$

$$|g_i(x) - y_i| \leq \epsilon, \quad i \in [n_y], \tag{11c}$$

where $\epsilon \geq 0$ a user-specified tolerance. Note that this relaxation of Problem (3) only affects the nonlinear functions g_i, $i \in [n_y]$.

4.1 Core ideas and notation

The main idea of the decomposition method is to relax the graphs of g_i with help of a set Ω_i. The set Ω_i is given by linear constraints built around g_i using the corresponding Lipschitz constant L_i. The resulting relaxation Ω_i is then refined in each iteration k such that Ω_i^k can be written as a union of polytopes

$$\Omega_i^k = \bigcup_{j \in J_i^k} \Omega_i^k(j),$$

which converges towards the graph of g_i for $k \to \infty$. The union of polytopes is uniquely defined in each iteration k by a set of points on the x-axis,

$$\mathcal{X}_i^k := \left\{ x_i^{k,0}, x_i^{k,1}, \ldots, x_i^{k,|J_i^k|} \right\},$$

where $x_i^{k,j} \in \mathbb{R}$ are scalar values for $j \in \{0\} \cup J_i^k = \{0\} \cup \{1, \ldots, |J_i^k|\}$ and

$$\underline{x}_i =: x_i^{k,0} < x_i^{k,1} < \cdots < x_i^{k,|J_i^k|} := \bar{x}_i.$$

Note that while x_i are scalar, the index i indicates that the sets $\mathcal{X}_i$ can develop differently from each other, specifically to every $i \in [n_y]$. In other words, the relaxations of g_i can be refined individually in each iteration k. This relaxation Ω_i^k of g_i is improved by adding a new point $x_i^{k,j}$ to $\mathcal{X}_i^k$. The polytopes of the union mentioned above are given by

$$\begin{aligned}\Omega_i^k(j) = \Big\{ (x, y_i) \in \mathbb{R}^2 \,:\, & x_i^{k,j-1} \le x \le x_i^{k,j}, \\ & y_i \le g_i\big(x_i^{k,j-1}\big) + L_i\big(x - x_i^{k,j-1}\big), \\ & y_i \ge g_i\big(x_i^{k,j-1}\big) - L_i\big(x - x_i^{k,j-1}\big), \\ & y_i \le g_i\big(x_i^{k,j}\big) + L_i\big(x_i^{k,j} - x\big), \\ & y_i \ge g_i\big(x_i^{k,j}\big) - L_i\big(x_i^{k,j} - x\big) \Big\}\end{aligned} \tag{12}$$

for $j \in J_i^k$. Visually speaking, (12) states that the polytopes $\Omega_i^k(j)$ are all quadrilaterals with two vertices $x_i^{k,j-1}$ and $x_i^{k,j}$ on the graph of g_i; see Fig. 1. To understand how a point is added to $\mathcal{X}_i$, we discuss the two problems solved in each iteration of the algorithm.

4.2 The master problem

The master problem (M(k)) is solved in each iteration over the sets Ω_i^k. The problem is formally given by

$$\begin{aligned} \min_{\omega} \quad & cx + d^\top y \\ \text{s.t.} \quad & Ax \ge a, \quad \underline{x} \le x \le \bar{x}, \quad x \in \mathbb{R}, \\ & \omega_i \in \Omega_i^k, \quad i \in [n_y], \end{aligned} \tag{M(k)}$$

with $\omega = (x, y)$ and $\omega_i := (x, y_i)$. Note that ω_i is not the ith entry in ω, since $\omega \in \mathbb{R}^{1+n_y}$ is the vector containing x and y while $\omega_i \in \mathbb{R}^2$ contains x and y_i. If the solution ω^k with $x^k \in \mathbb{R}$ and $y^k \in \mathbb{R}^{n_y}$ of (M(k)) is ε-feasible, then Problem (3) is approximately solved. According to Schmidt et al. [25], the master problem (M(k)) can be modeled as the mixed-integer linear problem

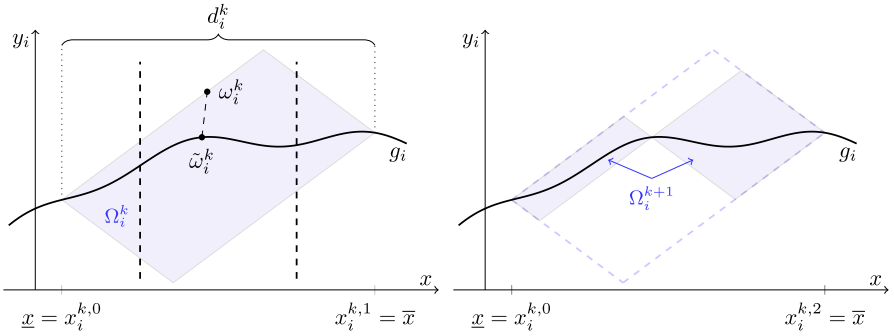


Fig. 1 Left: The subproblem is solved on the set $\widetilde{\Omega}_i^k$, which is bounded by the dashed, vertical lines. Right: The feasible set of the master problem in iteration $k+1$, after $\tilde{x}_i$ has been added to $\mathcal{X}_i^k$. The dashed lines depict the old feasible set

$$\min_{\omega} \quad cx + d^\top y \tag{13a}$$

$$\text{s.t.} \quad Ax \geq a, \quad \underline{x} \leq x \leq \bar{x}, \quad x \in \mathbb{R}, \tag{13b}$$

$$-M\big(1 - z_i^{k,j}\big) + x_i^{k,j-1} \leq x \leq x_i^{k,j} + M\big(1 - z_i^{k,j}\big), \quad i \in [n_y],\ j \in J_i^k, \tag{13c}$$

$$y_i \leq g_i\big(x_i^{k,j-1}\big) + L_i\big(x - x_i^{k,j-1}\big) + M\big(1 - z_i^{k,j}\big), \quad i \in [n_y],\ j \in J_i^k, \tag{13d}$$

$$y_i \geq g_i\big(x_i^{k,j-1}\big) - L_i\big(x - x_i^{k,j-1}\big) - M\big(1 - z_i^{k,j}\big), \quad i \in [n_y],\ j \in J_i^k, \tag{13e}$$

$$y_i \leq g_i\big(x_i^{k,j}\big) + L_i\big(x_i^{k,j} - x\big) + M\big(1 - z_i^{k,j}\big), \quad i \in [n_y],\ j \in J_i^k, \tag{13f}$$

$$y_i \geq g_i\big(x_i^{k,j}\big) - L_i\big(x_i^{k,j} - x\big) - M\big(1 - z_i^{k,j}\big), \quad i \in [n_y],\ j \in J_i^k, \tag{13h}$$

$$\sum_{j \in J_i^k} z_i^{k,j} = 1, \quad i \in [n_y], \tag{13g}$$

$$z_i^{k,j} \in \{0,1\}, \quad i \in [n_y],\ j \in J_i^k. \tag{13i}$$

The constraint $\omega_i \in \Omega_i^k$, $i \in [n_y]$, in (M(k)) is modeled here using big-M constraints. The binary variables z in (13i) and (13h) ensure that only one polytope is active for all $i \in [n_y]$.

4.3 The subproblem

On the other hand, if the solution ω^k is not ϵ-feasible, the relaxation Ω_i^k of the graph of g_i needs to be refined. This is achieved by solving a subproblem to find a point on the graph of g_i, which is as close as possible to the solution $\omega_i^k \in \Omega_i^k$ of the master problem (M(k)). We then use this point to refine the relaxation.

Our subproblem differs from the original method described in Schmidt et al. [25]. There, an optimization problem over nonlinearities and other linear constraints is solved. This is not possible in the setting we consider here, or is at least extremely challenging, due to the nonconvex and nonsmooth nature of neural networks.

The subproblem is solved only for the particular polytope $\Omega_i^k(j_i^k) \subset \Omega_i^k$ with $\omega_i^k \in \Omega_i^k(j_i^k)$. That is, we look for a point on the graph of g_i, which is contained in $\Omega_i^k(j_i^k)$. Additionally, the feasible set of the subproblem is further reduced to only allow for a solution in an inner-approximation of the respective polytope. This way we ensure that newly found points do not accumulate at an already existing value in $\mathcal{X}_i^k$. For a given $j \in J_i^k$, this subset is defined as

$$\widetilde{\Omega}_i^k(j) = \Omega_i^k(j) \cap \widehat{\Omega}_i^k(j),$$

with

$$\widehat{\Omega}_i^k(j) = \left\{ (x, y_i) \in \mathbb{R}^2 \,:\, x_i^{k,j-1} + \frac{1}{4} d_i^{k,j} \le x \le x_i^{k,j} - \frac{1}{4} d_i^{k,j} \right\}$$

and $d_i^{k,j} = x_i^{k,j} - x_i^{k,j-1}$ is the length of the corresponding segment on the x-axis. The constant 0.25 can be replaced by any value in (0, 0.5). The left plot in Fig. 1 indicates the set $\widehat{\Omega}_i^k$ with vertical dashed lines.

To solve the subproblem, we sample points $\tilde{x}_i$ on the x-axis segment corresponding to $\widetilde{\Omega}_i^k(j_i^k)$ and evaluate g_i at these points. Then, the solution of the subproblem is given by the computed point $\tilde{\omega}_i := (\tilde{x}_i, g_i(\tilde{x}_i))$ that is closest to the solution $\omega_i^k = (x^k, y_i^k)$ of the master problem w.r.t. the Euclidean distance; see Fig. 1. In other words, we choose from a finite set of points in $\widetilde{\Omega}_i^k(j_i^k)$ the one that is on the graph of g_i and as close as possible to the solution ω_i^k of (M(k)). Note that we solve the subproblem only for those $i \in [n_y]$ that are not ε-feasible, i.e., if $|g_i(x^k) - y_i^k| > \varepsilon$ holds.

The solution $\tilde{x}_i^k$ of the subproblem is added to the set $\mathcal{X}_i^k$ and thus refines the relaxation of g_i; see Fig. 1 (right). While x^k in ω_i^k is the solution of (M(k)) and thus shared by all $i \in [n_y]$, the solution $\tilde{x}_i^k$ of the subproblem unique to $i \in [n_y]$.

4.4 Algorithm and convergence properties

The Lipschitz decomposition method is formally given in Algorithm 1. There, the master problem (M(k)) is solved in each iteration k and the algorithm checks if the solution ω^k is ε-feasible. If this is not the case for all $i \in [n_y]$, the polytope $\Omega_i^k(j_i^k)$ containing the solution ω_i^k is identified using the index of the variables $z_i^{k,j}$ in the MILP (13) for all ε-infeasible $i \in [n_y]$. Then, a new point $\tilde{\omega}_i^k = (\tilde{x}_i^k, \tilde{y}_i^k)$ is found on a subset $\widetilde{\Omega}_i^k(j_i^k) \subset \Omega_i^k(j_i^k)$ and $\tilde{x}_i^k$ is added to $\mathcal{X}_i^k$, refining the approximation of g_i. Notice that while x is scalar, the index i in $\tilde{x}_i^k$ signals that the new point found on the x-axis in iteration k is specific to the relaxation of function g_i. Moreover, the number of points in $\mathcal{X}_i^k$ is at most k in every iteration, which means that $|J_i^k| \le k$ for all $i \in [n_y]$.

We remark that the new point $\tilde{x}_i^k$ found by the subproblem splits the original quadrilateral $\Omega_i^k(j_i^k)$ into two smaller ones; see Fig. 1. For the two new polytopes, we use the notations $\Omega_i^k(j_1^k)$ and $\Omega_i^k(j_2^k)$. Hence, the union of polytopes Ω_i^{k+1} that approximates g_i in iteration $k+1$ is given by

$$\Omega_i^{k+1} = \Omega_i^k(j_1^k) \bigcup \Omega_i^k(j_2^k) \bigcup_{j \ne j_i^k \in J_i^k} \Omega_i^k(j)$$

and we have the following lemma.

Algorithm 1 Lipschitz Decomposition Method

Input: Problem (3) and $\varepsilon > 0$
Output: An approximate globally optimal and ε-feasible point for Problem (3) or indication of infeasibility

1: Set $k \leftarrow 1$ and initialize $\mathcal{X}_i^k = \{\underline{x}, \bar{x}\}$ for all $i \in [n_y]$.
2: **while** true **do**
3: Solve the master problem (13) to global optimality.
4: **if** (13) is infeasible **then**
5: **return** "Problem (3) is infeasible."
6: **end if**
7: Let $\omega^k = (x^k, y^k)$ denote the optimal solution of (13).
8: **if** $|g_i(x^k) - y_i^k| \leq \varepsilon$ for all $i \in [n_y]$ **then**
9: **return** ω^k.
10: **end if**
11: Determine the polytopes $j_i^k \in J_i^k$ for all $i \in [n_y]$.
12: Solve the subproblems for all $i \in [n_y]$ with $|g_i(x^k) - y_i^k| > \varepsilon$ and let $\tilde{\omega}_i^k = (\tilde{x}_i^k, \tilde{y}_i^k)$ denote the optimal solution.
13: **for** $i \in [n_y]$ **do**
14: **if** $|g_i(x^k) - y_i^k| > \varepsilon$ **then**
15: Set $\mathcal{X}_i^{k+1} \leftarrow \mathcal{X}_i^k \cup \{\tilde{x}_i^k\}$.
16: **else**
17: Set $\mathcal{X}_i^{k+1} \leftarrow \{\mathcal{X}_i^k\}$.
18: **end if**
19: **end for**
20: Increase $k \leftarrow k + 1$.
21: **end while**

Lemma 3 (See Lemma 4 in Schmidt et al. [25]) *There exists a constant $\delta > 0$ depending only on ε and L such that as long as Algorithm 1 does not terminate in Line 5 or 9, there exists a constant $\delta^k > \delta$ for every k with*

$$\mathrm{Vol}\left(\Omega_i^k(j_1^k)\right) + \mathrm{Vol}\left(\Omega_i^k(j_2^k)\right) = \mathrm{Vol}\left(\Omega_i^k(j_i^k)\right) - \delta^k.$$

Theorem 4 (*See Theorem 1 in Schmidt et al. [25]*) *There exists a $K < \infty$ such that Algorithm 1 either terminates with an approximate globally optimal point ω^k or with the indication of infeasibility in an iteration $k \leq K$.*

Similarly to Theorem 1 in Schmidt et al. [25], Theorem 4 follows from the fact that, according to Lemma 3, the volume of Ω_i^k decreases by a positive value δ^k in each iteration that is uniformly bounded away from zero.

However, due to the fact that Ω_i is refined in each iteration to better approximate g_i, the master problem (M(k)) grows linearly over the course of the iterations, leading to one additional binary variable $z_i^{k,j}$ and some additional linear constraints. Consequently, the computational effort increases in every iteration.

5 Case study

In this section, we illustrate the applicability of Algorithm 1 on the basis of a set of exemplary case studies. We first discuss some implementation details and then present the results of the algorithm.

5.1 Implementation details

We now explain how we generate the (x, y)-pairs for the considered instances. We also discuss the used neural networks and their training, the sampling of points for the subproblem, and how we verify the solutions obtained with our algorithm.

We generate the (x, y)-pairs as follows. First, we solve the high-point relaxation of the bilevel problem but with a different objective function in which we either minimize or maximize the upper-level variable x to get the set $\mathcal{F}_x$. Then, we equidistantly sample in this interval and solve the parametric lower-level problem for the given values of x, obtain the lower-level problem's solution y, and store the point (x, y) with $y = \Psi(x)$ in our data set. The resulting data set entirely consists of bilevel feasible points. The obtained (x, y)-pairs are then used to train neural networks to learn the optimal follower's response. Note that using the modified high-point relaxation to obtain the x-interval used for sampling is not possible in reality if the lower-level problem is not known. However, a generation procedure would not be required in reality at all since the set of (x, y)-pairs for training the neural networks consist of observed data from the past.

The lower-level problem of every instance is then assumed to be unknown. Instead, for every considered problem, the optimal follower's response for a given leader's decision x, i.e., $\Psi_i(x)$, is learned with feed-forward neural networks with ReLU activation functions at every node, for all $i \in [n_y]$; see, e.g., Goodfellow et al. [13]. The functions learned with ReLU networks are inherently piecewise linear and, thus, Lipschitz continuous. Moreover, for all instances the weights of the network are updated via the Adam optimizer [24]. We vary the learning rate, the number of epochs, and the network architecture depending on the instance at hand.

The bilevel feasible set $\mathcal{F}$ and its projection $\mathcal{F}_x$ onto the x-variables also depend on the unknown lower level. Hence, we attempt to deduce $\mathcal{F}$ and $\mathcal{F}_x$ from the available (x, y)-pairs. Due to the fact that the interval $[\underline{x}, \bar{x}]$ with $\underline{x} \leq x \leq \bar{x}$ can be larger than $\mathcal{F}_x$, we do not use $\underline{x}$ and $\bar{x}$ to initialize Ω_i; see (12). Instead, we use the closed interval generated by the smallest and the largest x-values in the available training set as a proxy for $\mathcal{F}_x$.

Given the trained networks g_i, we solve the master problem and, subsequently, the subproblems as described in Sect. 4. To solve the subproblem, we evaluate the functions g_i at $p = 100$ points on the corresponding x-axis segment. If s points have already been evaluated on this segment in earlier iterations, then only $p - s$ new equidistantly distributed points $(x, g_i(x))$ are computed.

Finally, Algorithm 1 stops with the indication that Problem (3) is infeasible or with an ε-feasible solution, where $\varepsilon = 10^{-5}$ is used in our experiments. The solution

computed by our algorithm is compared with the solution obtained by solving the mixed-integer KKT reformulation with sufficiently large big-M constants; see, e.g., Kleinert et al. [18].

We implemented the LipSDP-Neuron method using the cvxpy 1.1.13 package and the SDP solver MOSEK 9.3.20. All occurring linear or mixed-integer linear problems (in particular, (M(k))) are solved using Gurobi 9.5.1. All neural networks have been trained using the Python library torch 1.11.0. All computations have been executed on a Intel© Core™i7-10510U CPU with 8 cores of 1.8 GHz each and 32 GB RAM.

5.2 Discussion of the results

We apply the Lipschitz decomposition method to 6 instances of linear bilevel problems from the literature. All instances have a scalar upper-level decision variable x so that Assumption 1 is satisfied. For all 6 instances we use 60 % of the (x, y)-pairs for training and the rest for validation. Alternative proportions of training and validation set sizes could be used as well if appropriate. Furthermore, given that $\Psi_i(\cdot), i \in [n_y]$, is piecewise linear in our context, rather small data sets are often already enough to find good solutions for the considered instances.

Table 1 shows the training set sizes, the configurations of the neural networks, and the used learning rates. Table 2 contains the corresponding Lipschitz constants computed for the networks described in the previous table as well as the time required to compute them. Finally, Table 3 displays the solutions obtained with Algorithm 1 (as well as the required number of iterations) next to the ones computed by solving the KKT reformulation. All computation times are given in seconds.

5.2.1 The performance of Algorithm 1

As discussed in Sect. 4.4, the computational effort of solving the master problem naturally increases over the course of the iterations. This can also be seen clearly in Fig. 2 for 4 exemplarily chosen instances. Furthermore, the computed Lipschitz constants determine the volume of the polytopes Ω_i^k, $i \in [n_y]$. Thus, according to Lemma 3 and Theorem 4, the convergence of Algorithm 1 directly depends on the constants L_i.

One way to keep Algorithm 1 most efficient is to compute Lipschitz constants L_i that are as small as possible. To illustrate this, we consider the instance [21]

$$\max_{x,y} \quad F(x, y) = -x - 2x \tag{14a}$$

$$\text{s.t.} \quad y \in \Psi(x), \tag{14b}$$

where $\Psi(x)$ is the set of optimal solutions of the x-parameterized linear lower-level problem

Table 1 Basic information about the instance-specific neural networks

Instance	Training Set size	Network Architecture	Learning Rate
Bard [1]	30	[1, 15, 10, 15, 1]	0.055
Bialas and Karwan [5]	30	[1, 20, 10, 20, 1]	0.049
Clark and Westerberg [8]	30	[1, 10, 15, 10, 1]	0.023
Haurie et al. [16]	30	[1, 10, 15, 10, 1]	0.049
Liu and Hart [19]	30	[1, 20, 10, 20, 1]	0.094
Moore and Bard [21]	30	[1, 5, 5, 1]	0.074

All networks are trained for 200 epochs and have training losses close to zero. The learning rates are rounded to three decimals

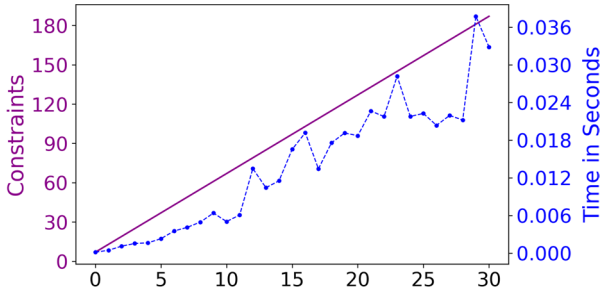
Table 2 Lipschitz constants as computed with the LipSDP-Neuron method using MOSEK 9.3.20; see Sect. 3

Instance	Lipschitz constant	Computation time
Bard [1]	5.77	0.049
Bialas and Karwan [5]	7.69	0.052
Clark and Westerberg [8]	0.98	0.045
Haurie et al. [16]	7.3	0.059
Liu and Hart [19]	21.65	0.081
Moore and Bard [21]	3.02	0.022

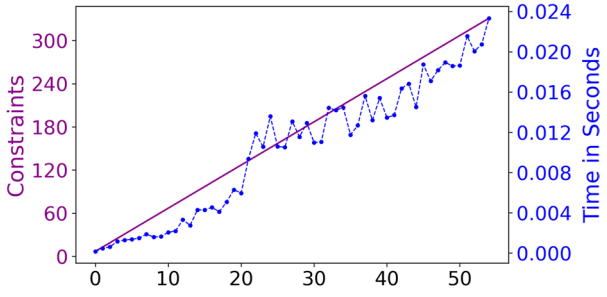
Table 3 Algorithm 1 solves the instances above using the networks in Table 1 and the corresponding Lipschitz constants in Table 2

Instance	Solution of KKT reform.	Solution of Algorithm 1	Nr. of Iterations	Comp. Time
Bard [1]	(7.2, 1.6)	(7.07, 1.54)	12	0.3
Bialas and Karwan [5]	(16, 11)	(15.67, 10.34)	55	2.23
Clark and Westerberg [8]	(19, 14)	(18.63, 13.76)	33	1.42
Haurie et al. [16]	(12, 3)	(11.79, 3.15)	49	4.44
Liu and Hart [19]	(4, 4)	(3.92, 3.67)	58	3.22
Moore and Bard [21]	(0, 1.5)	(0, 1.4999)	31	1.21

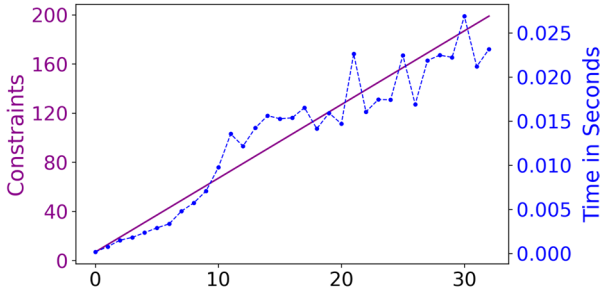
$$
\begin{aligned}
\max_{y} \quad & f(x,y) = y && \text{(15a)}\\
\text{s.t.} \quad & -x + 2.5y \leq 3.75, && \text{(15b)}\\
& x + 2.5y \geq 3.75, && \text{(15c)}\\
& 2.5x + y \leq 8.75, && \text{(15d)}\\
& x, y \geq 0, && \text{(15e)}\\
& x, y \in \mathbb{R}. && \text{(15f)}
\end{aligned}
$$



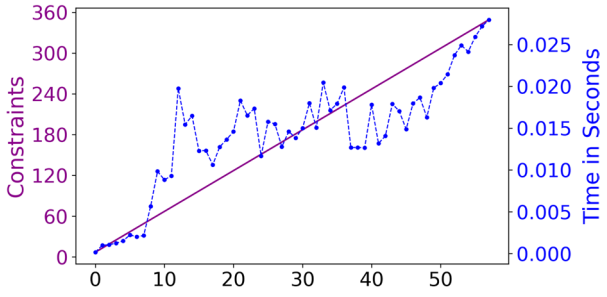
(A) Moore and Bard (1990)



(B) Bialas and Karwan (1984)



(C) Clark and Westerberg (1988)



(D) Liu and Hart (1994)

Fig. 2 The computational effort of Algorithm 1 grows linearly in the number of constraints in each iteration. The *x*-axis shows the number of iterations required by Algorithm 1

The optimal solution of Problem (14) is (0, 1.5). As explained in Sect. 5.1, we use the smallest and the largest x-values in the available data set to initialize $[\underline{x}, \bar{x}]$, which is required in Line 1 of Algorithm 1. For this instance, the unknown lower level is substituted by a neural network that has 2 hidden layers with 5 nodes each and is trained on a set of 30 points. The learning rate is approximately 0.074. Using the trained network's weights, we compute a Lipschitz constant of about 3.02. Finally, Algorithm 1 finds the approximate solution (0, 1.4999) in 1.21 seconds and 31 iterations for $\varepsilon = 10^{-5}$.

Table 4 captures how different Lipschitz constants influence the computation time and the number of iterations of Algorithm 1 when applied to Problem (14). For all computations, we keep the tolerance $\varepsilon = 10^{-5}$. This table clearly shows that the algorithm performs better, the closer the used Lipschitz constant is to the true value of 2.5.

Moreover, constants smaller than 2.5 seemingly perform even better. For instance, using 0.5, Algorithm 1 finds a solution in only 10 iterations. Nevertheless, using values smaller than the Lipschitz constant of function $g_i(\cdot)$ can potentially lead to false infeasibilities reported by the algorithm.

5.2.2 The accuracy of Algorithm 1

According to (11), an ε-feasible solution of an instance is ε-close to the graph of all functions $g_i(\cdot)$. This does not mean that it is also necessarily close to the true optimal response $\Psi_i(\cdot)$, $i \in [n_y]$. Consequently, the accuracy of the method depends on the ability of the neural network to approximate the true optimal responses as good as possible.

Figure 3 illustrates the importance of a good approximation. There you can see in the bottom plot that the ε-feasible solution, in this case (0.03, 1.67), obtained for instance (14), is ε-close to (0.03, g(0.03)) but not to (0.03, Ψ(0.03)).

In this context, it is also interesting that we observed that larger training sets can lead to better solutions. Table 5 shows how the computed Lipschitz constants and the solutions obtained for the instance in Bard [1] change with increasing data sets.

Table 4 Computation times in dependence of the Lipschitz constant

Lipschitz constant	Iterations	Computation time	Solution
5	65	3.67	(0, 1.4999)
3.02	31	1.08	(0, 1.4999)
2.5	33	1.06	(0, 1.5)
1.5	20	0.45	(0, 1.4999)
0.5	10	0.13	(0, 1.4999)
0.1	–	–	–

The second row corresponds to the Lipschitz constant obtained with LipSDP-Neuron, and the third one to the true Lipschitz constant of $\Psi(\cdot)$ for instance (14)

For this instance, the true solution is (7.2, 1.6). Due to the fact that the solution is located at an extreme point of $\mathcal{F}_x$, the accuracy of Algorithm 1 directly depends on the proxy for $\mathcal{F}_x$ in this specific example.

6 Conclusion

In many practical situations, the leader of a bilevel optimization problem is not aware of an explicit formulation of the follower's problem. To cope with this issue, we proposed a method that uses neural networks to learn the follower's optimal reaction from past bilevel solutions. After training of the network, we compute Lipschitz constants for the learned functions and use a Lipschitz decomposition method to solve the reformulated, single-level problem with neural-network constraints.

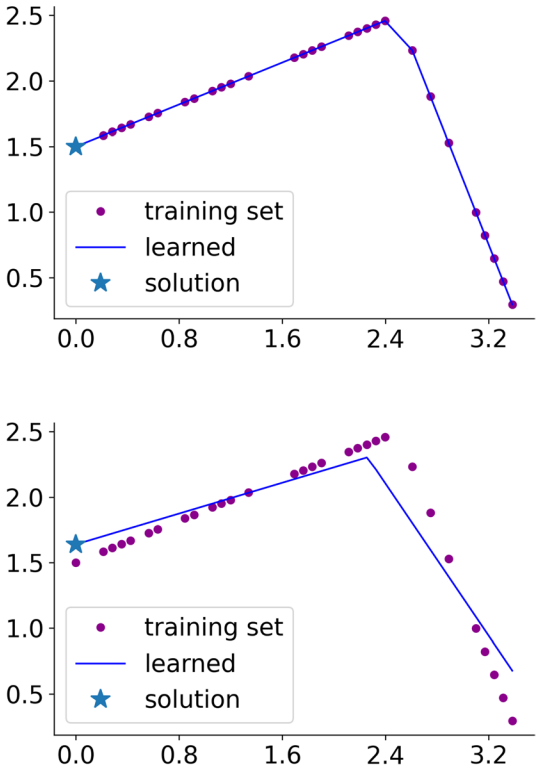


Fig. 3 Two neural networks trained on the same set with different learning rates. Consequently, the learned function g (blue curves) is different and Algorithm 1 computes two different solutions (blue stars) for instance (14) using these functions

Table 5 Lipschitz constants and solutions in dependence of the size of the training set for the instance in Bard [1]

Training set size	Lipschitz constant	Solution
6	5.04	(6.5, 1.25)
12	6.64	(6.87, 1.43)
30	5.77	(7.07, 1.54)
60	6.35	(7.01, 1.5)
120	8.43	(7.17, 1.58)

This short paper should serve as a proof of concept for the ideas sketched above. However, many aspects can be improved. First of all, the assumption of a scalar leader's decision seems rather strong. Very recently, a follow-up paper [14] of Schmidt et al. [25] appeared, in which a similar Lipschitz decomposition method is developed that can tackle the multi-dimensional case. This method can now be used to also consider bilevel optimization problems with an unknown follower problem and multiple decision variables of the leader. Second, we restricted ourselves in this paper to the case in which we have no coupling constraints. Fortunately, it is rather straightforward to use Algorithm 1 also for linear bilevel problems with coupling constraints as well. Even if the bilevel feasible set is disconnected due to the presence of coupling constraints, the follower's optimal response function $\Psi_i(\cdot)$ is learned to be a piecewise linear function by the corresponding neural network. Due to the fact that solutions are found at vertices of the feasible set, at least one feasible point is always available to serve as solution, even if the line connecting two points is not bilevel feasible. On the other hand, this is getting more complicated if nonlinear bilevel problems are considered. Hence, the setting of nonlinear problems with coupling constraints is a topic of future research. Third, the overall idea should be reasonable also in the mixed-integer case, at least if no coupling constraints are present. Fourth, there has been some recent work [6] on the relation between multilevel mixed-integer linear optimization problems and multistage stochastic mixed-integer linear optimization problems with recourse. Hence, it might be possible to exploit these relations to carry over learning-based techniques for two-stage stochastic optimization to bilevel optimization. Fifth and finally, it would be very interesting to see an application of the concept discussed in this paper to a real-world situation, which is, however, out the of the scope of this short paper.

Acknowledgements The authors thank the DFG for their support within RTG 2126 "Algorithmic Optimization".

Funding Open Access funding enabled and organized by Projekt DEAL.

References

1. Bard, J.F.: Optimality conditions for the bilevel programming problem. Naval Res. Logist. Q. **31**(1), 13–26 (1984). https://doi.org/10.1002/nav.3800310104
2. Beck, Y., Ljubic, I., Schmidt M.: A Survey on Bilevel Optimization Under Uncertainty. Technical report. http://www.optimization-online.org/DB_HTML/ 2022/06/8963.html (2022)
3. Besançon, M., Anjos, M. F., Brotcorne, L.: Near-optimal Robust Bilevel Optimization. arxiv:1908.04040pdf (2019)
4. Besançon, M., Anjos, M.F., Brotcorne, L.: Complexity of near-optimal robust versions of multilevel optimization problems. Optim. Lett. **15**, 2597–2610 (2021). https://doi.org/10.1007/s11590-021-01754-9
5. Bialas, W.F., Karwan, M.H.: Two-level linear programming. Manag. Sci. **30**(8), 1004–1020 (1984)
6. Bolusani, S., Coniglio, S., Ralphs, T.K., Tahernejad, S.: A unified framework for multistage mixed integer linear optimization. In: Dempe, S., Zemkoho, A. (eds.) Bilevel Optimization: Advances and Next Challenges, pp. 513–560. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-52119-6_18
7. Borrero, J.S., Prokopyev, O.A., Sauré, D.: Learning in sequential bilevel linear programming. In: INFORMS Journal on Optimization. https://doi.org/10.1287/ijoo.2021.0063 (2022)
8. Clark, P., Westerberg, A.: A note on the optimality conditions for the bilevel programming problem. In Naval Research Logistics (NRL) **35**(5), 413–418 (1988). https://doi.org/10.1002/1520-6750(198810)35:5<413::AID-NAV3220350505>3.0.CO;2-6
9. Dempe, S.: Foundations of Bilevel Programming. Springer, Berlin (2002)
10. Dempe, S.: Bilevel Optimization: theory, algorithms, applications and a bibliography. In: Dempe, S., Zemkoho, A. (eds.) Bilevel Optimization: Advances and Next Challenges, pp. 581–672. Springer, Berlin (2020). https://doi.org/10.1007/978-3-030-52119-6_20
11. Fazlyab, M., Robey, A., Hassani, H., Morari, M., Pappas, G.: Efficient and accurate estimation of lipschitz constants for deep neural networks. In: Advances in Neural Information Processing Systems. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett. Vol. 32. Curran Associates, Inc. (2019). https://proceedings.neurips.cc/paper/2019/file/95e1533eb1b20a97777749fb94fdb944-Paper.pdf
12. Fazlyab, M., Morari, M., Pappas, G.J.: Safety verification and robustness analysis of neural networks via quadratic constraints and semidefinite programming. IEEE Trans. Autom. Control **67**(1), 1–15 (2022). https://doi.org/10.1109/TAC.2020.3046193
13. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press. https://mitpress.mit.edu/books/deep-learning (2016)
14. Grübel, J., Krug, R., Schmidt, M., Wollner, W.: A successive linear relaxation method for MINLPs with multivariate lipschitz continuous nonlinearities with applications to bilevel optimization and gas transport. Technical report. arxiv:2208.06444 (2022)
15. Hansen, P., Jaumard, B., Savard, G.: New branch-and-bound rules for linear bilevel programming. SIAM J. Sci. Stat. Comput. **13**(5), 1194–1217 (1992). https://doi.org/10.1137/0913069
16. Haurie, A., Savard, G., White, D.: A note on: an efficient point algorithm for a linear two-stage optimization problem. Oper. Res. **38**(3), 553–555 (1990). https://doi.org/10.1287/opre.38.3.553
17. Khanduri, P., Zeng, S., Hong, M., Wai, H.-T., Wang, Z., Yang, Z.: A Near-Optimal Algorithm for Stochastic Bilevel Optimization via Double-Momentum. arxiv:2102.07367 (2021)
18. Kleinert, T., Labbé, M., Ljubic, I., Schmidt, M.: A survey on mixed- integer programming techniques in bilevel optimization. EURO J. Comput. Optim. **9**, 100007 (2021). https://doi.org/10.1016/j.ejco.2021.100007
19. Liu, Y.-H., Hart, S.M.: Characterizing an optimal solution to the linear bilevel programming problem. Eur. J. Oper. Res. **73**(1), 164–166 (1994). https://doi.org/10.1016/0377-2217(94)90155-4
20. Lozano, L., Smith, J.C.: A value-function-based exact approach for the bilevel mixed-integer programming problem. Oper. Res. **65**(3), 768–786 (2017). https://doi.org/10.1287/opre.2017.1589

21. Moore, J.T., Bard, J.F.: The mixed integer linear bilevel programming problem. Oper. Res. **38**(5), 911–921 (1990)
22. Pauli, P., Koch, A., Berberich, J., Kohler, P., Allgöwer, F.: Training robust neural networks using Lipschitz bounds. IEEE Control Syst. Lett. **6**, 121–126 (2022). https://doi.org/10.1109/LCSYS.2021.3050444
23. Rios, L.M., Sahinidis, N.V.: Derivative-free optimization: a review of algorithms and comparison of software implementations. J. Glob. Optim. **56**(3), 1247–1293 (2013). https://doi.org/10.1007/s10898-012-9951-y
24. Ruder, S.: An overview of gradient descent optimization algorithms (2016). arxiv:1609.04747
25. Schmidt, M., Sirvent, M., Wollner, W.: A decomposition method for MINLPs with Lipschitz continuous nonlinearities. Math. Program. **178**(1), 449–483 (2019). https://doi.org/10.1007/s10107-018-1309-x
26. Schmidt, M., Sirvent, M., Wollner, W.: The cost of not knowing enough: mixed-integer optimization with implicit lipschitz nonlinearities. In: Optimization Letters (2021). https://doi.org/10.1007/s11590-021-01827-9 (Forthcoming)
27. Vlah, D., Šepetanc, K., Pandžic, H.: Solving Bilevel Optimal Bidding Problems Using Deep Convolutional Neural Networks. (2022) https://doi.org/10.48550/ARXIV.2207.05825