A Gradient Method for Multilevel Optimization

Ryo Sato
The University of Tokyo

Mirai Tanaka
The Institute of Statistical Mathematics
RIKEN

Akiko Takeda
The University of Tokyo
RIKEN

Abstract

Although application examples of multilevel optimization have already been discussed since the 1990s, the development of solution methods was almost limited to bilevel cases due to the difficulty of the problem. In recent years, in machine learning, Franceschi et al. have proposed a method for solving bilevel optimization problems by replacing their lower-level problems with the T steepest descent update equations with some prechosen iteration number T. In this paper, we have developed a gradient-based algorithm for multilevel optimization with n levels based on their idea and proved that our reformulation asymptotically converges to the original multilevel problem. As far as we know, this is one of the first algorithms with some theoretical guarantee for multilevel optimization. Numerical experiments show that a trilevel hyperparameter learning model considering data poisoning produces more stable prediction results than an existing bilevel hyperparameter learning model in noisy data settings.

1 Introduction

Multilevel optimization When modeling real-world problems, there is often a hierarchy of decision-makers, and decisions are taken at different levels in the hierarchy. In this work, we consider multilevel optimization problems whose simplest form[1] is the following:

$$\begin{aligned}\min_{x_1 \in S_1 \subseteq \mathbb{R}^{d_1}, x_2^*, \ldots, x_n^*} \; & f_1(x_1, x_2^*, \ldots, x_n^*) \text{ s.t.} \\ & x_2^* = \operatorname*{argmin}_{x_2 \in \mathbb{R}^{d_2}, x_3^*, \ldots, x_n^*} \; f_2(x_1, x_2, x_3^*, \ldots, x_n^*) \text{ s.t.} \\ & \qquad \ddots \\ & \qquad\qquad x_n^* = \operatorname*{argmin}_{x_n \in \mathbb{R}^{d_n}} f_n(x_1, x_2, \ldots, x_n), \end{aligned} \tag{1}$$

where d_i is positive integers, $x_i \in \mathbb{R}^{d_i}$ is the decision variable at the ith level, and $f_i \colon \mathbb{R}^{d_i} \to \mathbb{R}$ is the objective function of the ith level for $i = 1, \ldots, n$. In the formulation, an optimization problem contains another optimization problem as a constraint. The framework can be used to formulate problems in which decisions are made in sequence and earlier decisions influence later decisions. An optimization problem in which this structure is n-fold is called an n-level optimization problem or a multilevel optimization problem especially when $n \geq 3$.

Existing study on bilevel optimization Especially, when $n = 2$ in (1), it is called a bilevel optimization problem. Bilevel optimization or multilevel optimization has been a well-known problem in the field of optimization since the 1980s or 1990s respectively (see the survey paper [24] for the research at that time), and their solution methods have been studied mainly on bilevel

[1]This assumes that lower-level problems have unique optimal solutions $x_2^*, \ldots, x_n^*$ for the simplicity.

optimization until now. Recently, studies on bilevel optimization have received significant attention from the machine learning community due to practical applications and the development of efficient algorithms. For example, bilevel optimization was used to formulate decision-making problems for players in conflict [25, 22], and to formulate hyperparameter learning problems in machine learning [5, 11, 12]. Franceschi et al. [11] constructed a single-level problem that approximates a bilevel problem by replacing the lower-level problem with T steepest descent update equations using a prechosen iteration number T. It is shown that the optimization problem with T steepest descent update equations asymptotically converges to the original bilevel problem as $T \to \infty$. It is quite different from the ordinary approach (e.g., [1]) that transforms bilevel optimization problems into single-level problems using the optimality condition of the lower-level problem. The application of the steepest descent method to bilevel optimization problems has recently attracted attention, resulting in a stream of papers; e.g., [13, 20, 18].

Existing study on multilevel optimization Little research has been done on decision-making models (1) that assume the multilevel hierarchy, though various applications of multilevel optimization have been already discussed in the 1990s [24]. As far as we know, few solution methods with a theoretical guarantee have been proposed. A metaheuristic algorithm based on a perturbation, i.e., a solution method in which the neighborhood is randomly searched for a better solution in the order of x_1 to x_n has been proposed in [23] for general n-level optimization problems. The effect of updating x_i on the decision variables $x_{i+1}, \ldots, x_n$ of the lower-level problems is not considered, and hence, the hierarchical structure of the multilevel optimization problem cannot be fully utilized in updating the solution. Because of this, the algorithm does not have a theoretical guarantee for the obtained solution. Very recently (two days before the NeurIPS 2021 abstract deadline), a proximal gradient method based on the fixed-point theory for trilevel optimization problem [21] has been proposed. This paper has proved convergence to an optimal solution assuming the convexity of the objective functions. However, it does not include numerical results and hence its practical efficiency is unclear.

Contribution of this paper In this paper, by extending the gradient method for bilevel optimization problems [11] to multilevel optimization problems, we propose an algorithm with a convergence guarantee for multilevel optimization other than one by [21]. Increasing the problem hierarchy from two to three drastically makes developing algorithms and showing theoretical guarantees difficult. In fact, as discussed above, in the case of bilevel, replacing the second-level optimization problem with its optimality condition or replacing it with T steepest-descent sequential updates immediately results in a one-level optimization problem. However, when it comes to $n \geq 3$ levels, it may be necessary to perform this replacement n times, and it seems not easy to construct a solution method with a convergence guarantee. It is not straightforward at all to give our method a theoretical guarantee similar to the one [11] developed for bilevel optimization. We have confirmed the effectiveness of the proposed method by applying it to hyperparameter learning with real data. Experimental verifications of the effectiveness of trilevel optimization using real-world problems are the first ones as far as we know.

2 Related existing methods

2.1 Two-stage robust optimization problems

When making long-term decisions, it is also necessary to make many decisions according to the situation. If the same objective function is acceptable throughout the period, i.e., $f_1 = \cdots = f_n$, and there are hostile players, such a problem is often formulated as a multistage robust optimization problem. Especially, the concept of two-stage robust optimization (so-called adjustable robust optimization) $\min_{x_1 \in S_1} \max_{x_2 \in S_2} \min_{x_3 \in S_3} f_1(x_1, x_2, x_3)$ was introduced with feasible sets $S_i \subseteq \mathbb{R}^{d_i}$ for $i = 1, 2, 3$, and methodology based on affine decision rules was developed by [4]. Researches based on affine policies are still being actively conducted; see, e.g., [6, 26]. The two-stage robust optimization can be considered as a special case of (1) with $n = 3$ when constraints $x_2 \in S_2 \subseteq \mathbb{R}^{d_2}$ and $x_3 \in S_3 \subseteq \mathbb{R}^{d_3}$ are added. This approach is unlikely to be applicable to our problem (1) without strong assumptions such as affine decision rules.

There is another research stream on min-max-min problems including integer constraints (see e.g., [7, 25]). They transform the problem into the min-max problem by taking the dual for the inner "min" and apply cut-generating approaches or Benders decomposition techniques with the property of integer variables. While the approach is popular in the application studies of electric grid planning, it is restricted to the specific min-max-min problems and no more applicable to multilevel optimization problems (1).

2.2 Existing methods for bilevel optimization problems

Various applications are known for bilevel optimization problems, i.e., the case of $n = 2$ in (1):

$$\min_{x_1 \in S_1} f_1(x_1, x_2) \text{ s.t. } x_2 = \operatorname*{argmin}_{x_2'} f_2(x_1, x_2'). \tag{2}$$

One of the most well-known applications in machine learning is hyperparameter learning. For example, the problem that finds the best hyperparameter value in the ridge regression is formulated as

$$\min_{\lambda \geq 0} \|y_{\text{valid}} - X_{\text{valid}}\theta\|_2^2 \text{ s.t. } \theta = \operatorname*{argmin}_{\theta'} \|y_{\text{train}} - X_{\text{train}}\theta'\|_2^2 + \lambda\|\theta'\|_2^2,$$

where $(X_{\text{train}}, y_{\text{train}})$ are training samples, $(X_{\text{valid}}, y_{\text{valid}})$ are validation samples, and λ is a hyperparameter that is optimized in this problem. Under Assumption 1 restricted to $n = 2$ shown later, the existence of an optimal solution to the bilevel problem (2) is ensured by [12, Theorem 3.1].

There are mainly two approaches to solve bilevel optimization problems. The old practice is to replace the lower-level problem with its optimality condition and to solve the resulting single-level problem. It seems difficult to use this approach to multilevel problems because we need to apply the replacement n-times. The other one is to use gradient-based methods developed by Franceschi et al. [11, 12] for bilevel optimization problems. Their approach reduces (2) to a single-level problem by replacing the lower-level problem with T equations using a prechosen number T.

Hereinafter, we briefly summarize the results of Franceschi et al. [11, 12] for bilevel optimization. Under the continuous differentiability assumption for f_2, x_2 is iteratively updated by $x_2^{(t)} = \Phi^{(t)}(x_1, x_2^{(t-1)})$ at the tth iteration using an iterative method, e.g., the gradient descent method:

$$\Phi^{(t)}(x_1, x_2^{(t-1)}) = x_2^{(t-1)} - \alpha^{(t)}\nabla_{x_2} f_2(x_1, x_2^{(t-1)}).$$

Then, the bilevel optimization problem is approximated by the single-optimization problem with T equality constraints and new variables $\{x_2^{(t)}\}_{t=1}^T$ instead of x_2:

$$\min_{x_1 \in S_1, \{x_2^{(t)}\}} f_1(x_1, x_2^{(T)}) \text{ s.t. } x_2^{(t)} = \Phi^{(t)}(x_1, x_2^{(t-1)}) \ (t = 1, \ldots, T), \tag{3}$$

where $x_2^{(0)}$ is a given constant. Eliminating $x_2^{(1)}, \ldots, x_2^{(T)}$ using constraints, we can equivalently recast the problem above into the unconstrained problem $\min_{x_1 \in S_1} \tilde{F}_1(x_1)$.

Under assumption on differentiability, the gradient of $\tilde{F}_1(x_1)$ is given in [11, Section 3] by

$$\begin{aligned}
\nabla_{x_1}\tilde{F}_1(x_1) &= \nabla_{x_1} f_1(x_1, x_2^{(T)}) + \sum_{t=1}^{T} B^{(t)} \left(\prod_{s=t+1}^{T} A^{(s)} \right) \nabla_{x_2} f_1(x_1, x_2^{(T)}), \\
A^{(t)} &= \nabla_{x_2}\Phi^{(t)}(x_1, x_2^{(t-1)}) \ (t = 1, \ldots, T), \\
B^{(t)} &= \nabla_{x_1}\Phi^{(t)}(x_1, x_2^{(t-1)}) \ (t = 1, \ldots, T).
\end{aligned}$$

Roughly speaking, [12, Theorem 3.2] proved that the optimal value and the solution set of (3) converge to those of (2) as $T \to \infty$ under Assumptions 1 and 3 restricted to $n = 2$.

3 Multilevel optimization problems and their approximation

We will develop an algorithm for multilevel optimization problems (1) under some assumptions by extending the studies [11, 12] on bilevel problems. Our algorithm and its theoretical guarantee look similar to those in [11, 12], but they are not straightforwardly obtained. As emphasized in Section 1, unlike the bilevel problem, even if the lower-level problem is replaced with T steepest-descent sequential updates, the resulting problem is still a multilevel problem. In this section, we show how to resolve these difficulties that come from the multilevel hierarchy.

3.1 Existence of optima of multilevel optimization problems

First, we discuss the existence of an optimal solution of the multilevel optimization problem (1). To do so, we introduce the following assumption, which is a natural extension of that in [12].

Assumption 1.

(i) S_1 is compact.

(ii) For $i = 1, \dots, n$, f_i is jointly continuous.

(iii) For $i = 2, \dots, n$, the set of optimal solutions of the ith level problem with arbitrarily fixed $(x_1, \dots, x_{i-1})$ is a singleton.

(iv) For $i = 2, \dots, n$, the optimal solution of the ith level problem remains bounded as x_1 varies in S_1.

Assumption 1-(iii) means that the ith level problem with parameter $(x_1, \dots, x_{i-1})$ has a unique optimizer, that is described as x_i^* in (1) though it should be formally written as $x_i^*(x_1, \dots, x_{i-1})$ since it is determined by $x_1, \dots, x_{i-1}$. We define $F_i(x_1, \dots, x_i) := f_i(x_1, \dots, x_i, x_{i+1}^*, \dots, x_n^*)$ by eliminating $x_{i+1}^*, \dots, x_n^*$ since it depends only on $(x_1, \dots, x_i)$. Then, the ith level problem with parameter $(x_1, \dots, x_{i-1})$ can be written as $\min_{x_i} F_i(x_1, \dots, x_i)$. Similarly, in the 1st level problem, for fixed x_1, the remaining variables $x_2, x_3, \dots, x_n$ can be represented as $x_2^*(x_1)$, $x_3^*(x_1, x_2^*(x_1))$ and so on. In what follows, we denote them by $x_2^*(x_1), x_3^*(x_1), \dots, x_n^*(x_1)$ since they are determined by x_1. Eliminating them, we can rewrite the 1st level problem, equivalently (1), as

$$\min_{x_1 \in S_1} F_1(x_1). \tag{4}$$

Theorem 2. *Under Assumption 1, Problem* (4) *admits optimal solutions.*

Theorem 2 is a generalization of [12, Theorem 3.1], which is valid only for $n = 2$, to general n. It is difficult to extend the original proof to general n because, to derive the continuity of x_i^* extending the original proof, a sequence $(x_1, \dots, x_{i-1})$ approaches to its accumulation point only from a specific direction. Instead, in the multilevel case, we employ the theory of point-to-set mapping. A proof of Theorem 2 is shown in Supplementary material A.1.

3.2 Approximation by iterative methods

We extend the gradient method for bilevel optimization problems proposed by Franceschi et al. [11] to multilevel optimization problems (1). Based on an argument similar to one in Subsection 2.2, we approximate lower level problems in (1) by applying an iterative method with T_i iterations to the ith level problem for $i = 2, \dots, n$. Then, we obtain the following approximated problem:

$$\begin{aligned} &\min_{x_1 \in S_1, \{x_2^{(t_2)}\}, \dots, \{x_n^{(t_n)}\}} f_1(x_1, x_2^{(T_2)}, \dots, x_n^{(T_n)}) \\ &\qquad \text{s.t. } x_i^{(t_i)} = \Phi_i^{(t_i)}(x_1, x_2^{(T_2)}, \dots, x_{i-1}^{(T_{i-1})}, x_i^{(t_i-1)}) \ (i = 2, \dots, n; t_i = 1, \dots, T_i), \end{aligned} \tag{5}$$

where $\Phi_i^{(t_i)}(x_1, x_2^{(T_2)}, \dots, x_{i-1}^{(T_{i-1})}, x_i^{(t_i)})$ is the t_ith iteration formula of an iterative method for the ith level approximated problem with parameters $(x_1, \dots, x_{i-1}) = (x_1, x_2^{(T_2)}, \dots, x_{i-1}^{(T_{i-1})})$, i.e.,

$$\begin{aligned} &\min_{x_i, \{x_{i+1}^{(t_{i+1})}\}, \dots, \{x_n^{(t_n)}\}} f_i(x_1, \dots, x_i, x_{i+1}^{(T_{i+1})}, \dots, x_n^{(T_n)}) \\ &\qquad \text{s.t. } x_j^{(t_j)} = \Phi_j^{(t_j)}(x_1, \dots, x_i, x_{i+1}^{(T_{i+1})}, \dots, x_{j-1}^{(T_{j-1})}, x_j^{(t_j-1)}) \ (j = i+1, \dots, n; t_j = 1, \dots, T_j), \end{aligned}$$

and its initial point $x_i^{(0)}$ for each i are regarded as a given constant. For $i = 1, \dots, n$, if we fix $x_1, \dots, x_i$, each remaining variable $x_j^{(t)}$ is determined by constraints and thus, we denote it by $x_j^{(t_j)}(x_1, \dots, x_i)$ for $j = i+1, \dots, n$. Using this notation, the ith level objective function can be written as

$$\begin{aligned} &\tilde{F}_i(x_1, \dots, x_i) \\ &:= f_i(x_1, \dots, x_{i-1}, x_i, x_{i+1}^{(T_{i+1})}(x_1, \dots, x_i), \dots, x_n^{(T_n)}(x_1, \dots, x_{n-1}^{(T_{n-1})}(\cdots(x_{i+1}^{(T_{i+1})}(x_1, \dots, x_i))\cdots))), \end{aligned}$$

where $(x_1, \dots, x_{i-1})$ acts as a fixed parameter in the ith level problem. When we approximate the ith level problem with the steepest descent method for example, we use $\Phi_i^{(t_i)}(x_1, \dots, x_{i-1}, x_i^{(t_i)}) = x_i^{(t_i-1)} - \alpha_i^{(t_i-1)} \nabla_{x_i} \tilde{F}_i(x_1, \dots, x_{i-1}, x_i^{(t_i-1)})$.

Especially, all variables other than x_1 in (5) can be expressed as $\{x_2^{(t_2)}(x_1)\}, \dots, \{x_n^{(t_n)}(x_1)\}$ since it is determined by x_1. By plugging the constraints into the objective function and eliminating them, we can reformulate (5) as

$$\min_{x_1 \in S_1} \tilde{F}_1(x_1). \tag{6}$$

In the remainder of this subsection, we assume $T_2 = \dots = T_n = T$ for simplicity. Then, the optimal value and solutions of Problem (6) converge to those of Problem (4) as $T \to \infty$ in some sense. To derive it, we introduce the following assumption, which is also a natural extension of that in [12].

Assumption 3.

(i) $f_1(x_1, \cdot, \dots, \cdot)$ is uniformly Lipschitz continuous on S_1.

(ii) For all $i = 2, \dots, n$, sequence $\{x_i^{(T)}(x_1)\}$ converges uniformly to $x_i^*(x_1)$ on S_1 as $T \to \infty$.

Theorem 4. *Under Assumptions 1 and 3, the followings hold:*

(a) The optimal value of Problem (6) *converges to that of Problem* (4) *as $T \to \infty$.*

(b) The set of the optimal solutions of Problem (6) *converges to that of Problem* (4)*; more precisely, denoting an optimal solution of Problem* (6) *by $x_{1,T}^*$, we have*

- *$\{x_{1,T}^*\}_{T=1}^{\infty}$ admits a convergent subsequence;*
- *for every subsequence $\{x_{1,T_k}^*\}_{k=1}^{\infty}$ such that $x_{1,T_k}^* \to x_1^*$ as $k \to \infty$, the accumulation point x_1^* is an optimal solution of Problem* (4)*.*

See Supplementary material A.2 for a proof of this theorem.

4 Proposed method: Gradient computation in the approximated problem

Now we propose to apply a projected gradient method to the approximated problem (6) for multilevel optimization problems (1) because (6) asymptotically converges to (1) as shown in Theorem 4. In this section, we derive the formula of $\nabla_{x_1} \tilde{F}_1(x_1)$ and confirm the local and global convergence of the resulting projected gradient method.

4.1 Gradient of the objective function in the approximated problem

The following theorem provides a computation formula of $\nabla_{x_1} \tilde{F}_1(x_1)$.

Theorem 5 (Gradient formula for the n-level optimization problems)**.** *The gradient $\nabla_{x_1} \tilde{F}_1(x_1)$ can be expressed as follows:*

$$\begin{aligned}
\nabla_{x_1} \tilde{F}_1(x_1) &= \nabla_{x_1} f_1(x_1, x_2^{(T_2)}, \dots, x_n^{(T_n)}) + \sum_{i=2}^{n} Z_i \nabla_{x_i} f_1(x_1, x_2^{(T_2)}, \dots, x_n^{(T_n)}), \\
Z_i &= \sum_{t=1}^{T_i} \left(\sum_{j=2}^{i-1} Z_j C_{ij}^{(t)} + B_i^{(t)} \right) \prod_{s=t+1}^{T_i} A_i^{(s)}, \\
A_i^{(t)} &= \nabla_{x_i} \Phi_i^{(t)}(x_1, x_2^{(T_2)}, \dots, x_{i-1}^{(T_{i-1})}, x_i^{(t-1)}), \\
B_i^{(t)} &= \nabla_{x_1} \Phi_i^{(t)}(x_1, x_2^{(T_2)}, \dots, x_{i-1}^{(T_{i-1})}, x_i^{(t-1)}), \\
C_{ij}^{(t)} &= \nabla_{x_j} \Phi_i^{(t)}(x_1, x_2^{(T_2)}, \dots, x_{i-1}^{(T_{i-1})}, x_i^{(t-1)})
\end{aligned}$$

for any $i = 2, \dots, n$; $t = 1, \dots, T_i$; and $j = 2, \dots, i-1$, where we define $\prod_{s=t+1}^{T_i} A_i^{(s)} := A_i^{(t+1)} A_i^{(t+2)} \dots A_i^{(T_i)}$ for $t < T_i$ and $\prod_{s=T_i+1}^{T_i} A_i^{(s)} = I$.

See supplementary material A.3 for a proof of this theorem.

We consider computing $\nabla_{x_1}\tilde{F}_1(x_1)$ using Theorem 5. Notice that we can easily compute $Z_2 = \sum_{t=1}^{T_2} B_2^{(t)} \prod_{s=t+1}^{T_2} A_2^{(s)}$. For $i = 3, \dots, n$, when we have $Z_2, \dots, Z_{i-1}$, we can compute Z_i. We show an algorithm that computes $\nabla_{x_1}\tilde{F}_1(x_1)$ by computing $Z_2, \dots, Z_n$ in this order in Algorithm 1.

Algorithm 1 Computation of $\nabla_{x_1}\tilde{F}_1(x_1)$

```
Input: $x_1$: current value of the 1st level variable. $\{x_i^{(0)}\}_{i=2}^n$: initial values of the lower level iteration.
Output: The exact value of $\nabla_{x_1}\tilde{F}_1(x_1)$.
 1: $g \coloneqq (0, \dots, 0)^\top$.
 2: for $i \coloneqq 2, \dots, n$ do
 3:     $Z_i \coloneqq O$.
 4:     for $t \coloneqq 1, \dots, T_i$ do
 5:         $x_i^{(t)} \coloneqq \Phi_i^{(t)}(x_1, x_2^{(T_2)}, \dots, x_{i-1}^{(T_{i-1})}, x_i^{(t-1)})$.
 6:         $\bar{B}_i^{(t)} \coloneqq \sum_{l=2}^{i-1} Z_l C_{il}^{(t)} + B_i^{(t)}$.
 7:         $Z_i \coloneqq Z_i A_i^{(t)} + \bar{B}_i^{(t)}$.
 8: for $i = 2, \dots, n$ do
 9:     $g \coloneqq g + Z_i \nabla_{x_i} f_1$.
10: $g \coloneqq g + \nabla_{x_1} f_1$.
11: return $g$
```

For $i = 2, \dots, n$ and $t = 1, \dots, T_i$, $\Phi_i^{(t)}$, which appears in the 5th line of Algorithm 1, is the update formula based on the gradient $\nabla_{x_i}\tilde{F}_i(x_1, \dots, x_i)$ of the ith level objective function. $\nabla_{x_i}\tilde{F}_i(x_1, \dots, x_i)$ can be computed by applying Algorithm 1 to the $(n-i+1)$-level optimization problem with objective functions $\tilde{F}_i, \dots, \tilde{F}_n$. Therefore, recursively calling Algorithm 1 in the computation of $\Phi_i^{(t)}$, we can compute $\nabla_{x_1}\tilde{F}_1(x_1)$. For an example of applying Algorithm 1 to Problem (6) arising from a trilevel optimization problem, i.e., Problem (1) with $n = 3$, see Supplementary material B.

4.2 Complexity of the gradient computation

We analyze the complexity for computing $\nabla_{x_1}\tilde{F}_1(x_1)$ by recursively calling Algorithm 1. In the following theorem, the asymptotic big O notation is denoted by $\mathrm{O}(\cdot)$.

Theorem 6. *Let the time and space complexity for computing $\nabla_{x_i}\tilde{F}_i(x_i)$ be c_i and s_i, respectively. We use $\Phi_i^{(t_i)}$ based on $\nabla_{x_i}\tilde{F}_i(x_i)$ and recursively call Algorithm 1 for computing $\nabla_{x_i}\tilde{F}_i(x_1, \dots, x_i)$. In addition, we assume the followings:*

- *The time and space complexity for evaluating $\Phi_i^{(t_i)}$ are $\mathrm{O}(c_i)$ and $\mathrm{O}(s_i)$.*
- *The time and space complexity of $\nabla_{x_1} f_1$ and $\nabla_{x_i} f_1$ for $i = 1, \dots, n$ are smaller in the sense of the order than those of `for` loops in lines 2–9 in Algorithm 1.*

Then, the overall time complexity c_1 and space complexity s_1 for computing $\nabla_{x_i}\tilde{F}_i(x_1, \dots, x_i)$ can be written as

$$c_1 = \mathrm{O}\left(p^n n! c_n \prod_{i=1}^{n-1} (T_{i+1} d_i)\right), \quad s_1 = \mathrm{O}(q^n s_n), \tag{7}$$

respectively, for some constant $p, q > 1$.

For a proof, see Supplementary material A.4. Note that, if n is a fixed parameter, those complexity reduces to a polynomial of T_i's, d_i's, c_n, and s_n. Hence, Algorithm 1 can be regarded as a fixed-parameter tractable algorithm.

4.3 Global convergence of the projected gradient method

Here, we consider solving Problem (6) by the projected gradient method, which calculates the gradient vector by Algorithm 1 and projects the updated point on S_1 in each iteration. When all lower-level updates are based on the steepest descent method, we can derive the Lipschitz continuity of the gradient of the objective function of (6). Hence, we can guarantee the local and global convergence of the projected gradient method for (6) by taking a sufficiently small step size.

Theorem 7. *Suppose* $\Phi_i^{(t)}(x_1, \ldots, x_{i-1}, x_i^{(t-1)}) = x_i^{(t-1)} - \alpha_i^{(t-1)} \nabla_{x_i} \tilde{F}_i(x_1, \ldots, x_{i-1}, x_i^{(t-1)})$ *for all* $i = 2, \ldots, n$ *and* $t_i = 1, \ldots, T_i$*, where* $\alpha_i^{(t-1)}$ *and* $x_i^{(0)}$ *are given parameters for all* i *and* t*. Assume that* $\nabla_{x_j} f_i$ *is Lipschitz continuous and bounded for all* $i = 1, \ldots, n$ *and* $j = 1, \ldots, n$*; and also* $\nabla_{x_i} \Phi_i^{(t)}$*,* $\nabla_{x_1} \Phi_i^{(t)}$*, and* $\nabla_{x_j} \Phi_i^{(t)}$ *are Lipschitz continuous and bounded for all* $i = 2, \ldots, n$*;* $j = 2, \ldots, i-1$*;* $t = 1, \ldots, T_i$*. Then,* $\nabla_{x_1} \tilde{F}_1$ *is Lipschitz continuous.*

See Supplementary material A.5 for a proof of this theorem.

Corollary 8. *Suppose the same assumption as Theorem 7. Assume* S_1 *is a compact convex set. Let* L *be the Lipschitz constant of* $\nabla_{x_1} \tilde{F}_1$*. Then, a sequence* $\{x_1^{(t)}\}$ *generated by the projected gradient method with sufficiently small constant step size, e.g., smaller than* $2/L$*, for Problem* (6) *from any initial point has a convergent subsequence that converges to a stationary point with convergence rate* $\mathrm{O}(1/\sqrt{t})$*.*

Proof. From Theorem 7, the gradient of the objective function of Problem (6) is L-Lipschitz continuous. Let $G\colon \operatorname{int}(\operatorname{dom}(\tilde{F}_1)) \to \mathbb{R}^{d_1}$ be the gradient mapping [3, Definition 10.5] corresponding to $\tilde{F}_1$, the indicator function of S_1, and the constant step size $\alpha_1^{(t)}$ with satisfying $0 < \alpha_1^{(t)} < 2/L$ for all t. Note that $\|G(x_1)\| = 0$ if and only if x_1 is a stationary point of Problem (6) [3, Theorem 10.7]. By applying [3, Theorem 10.15], we obtain $\min_{s=0}^{t} \|G(x_1^{(s)})\| \leq \mathrm{O}(1/\sqrt{t})$ and $\|G(\bar{x}_1)\| = 0$, where $\bar{x}_1$ is a limit point of $\{x_1^{(t)}\}$. □

5 Numerical experiments

To validate the effectiveness of our proposed method, we conducted numerical experiments on an artificial problem and a hyperparameter optimization problem arising from real data (see Supplementary material C for complete results). In our numerical experiments, we implemented all codes with Python 3.9.2 and JAX 0.2.10 for automatic differentiation and executed them on a computer with 12 cores of Intel Core i7-7800X CPU 3.50 GHz, 64 GB RAM, Ubuntu OS 20.04.2 LTS.

In this section, we used Algorithm 1 to calculate the gradient of $\tilde{F}_i$ in problem 5, and used automatic differentiation to calculate the gradient of $\Phi_i^{(t)}$ in problem 5.

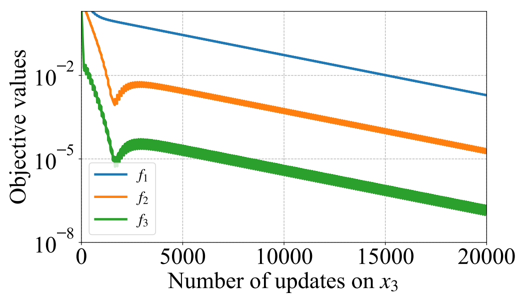
5.1 Convergence to the optimal solution

We solved the following trilevel optimization problem with Algorithm 1 to evaluate the performance:

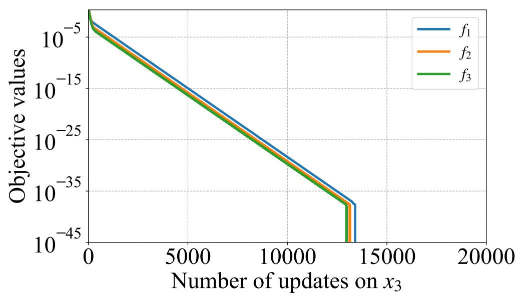
$$\begin{aligned} \min_{x_1 \in \mathbb{R}^2} f_1(x_1, x_2^*, x_3^*) &= \|x_3^* - x_1\|_2^2 + \|x_1\|_2^2 \text{ s.t.} \\ x_2^* \in \operatorname*{argmin}_{x_2 \in \mathbb{R}^2} f_2(x_1, x_2, x_3^*) &= \|x_2 - x_1\|_2^2 \text{ s.t.} \\ x_3^* \in \operatorname*{argmin}_{x_3 \in \mathbb{R}^2} f_3(x_1, x_2, x_3) &= \|x_3 - x_2\|_2^2. \end{aligned} \tag{8}$$

Clearly, the optimal solution for this problem is $x_1 = x_2 = x_3 = (0,0)^\top$.

We solved (8) with fixed constant step size and initialization but different (T_2, T_3). For the iterative method in Algorithm 1, we employed the steepest descent method at all levels. We show the transition of the value of the objective functions in Figure 1 and the trajectories of each decision variable in Figure 2. Since updates of x_3 is the most inner iteration, the time required to update x_3 does not change when T_2 or T_3 changes. Hence the number of updates of x_3 is proportional to the total computational time. Therefore, we can compare the time efficiency of the optimization algorithm by

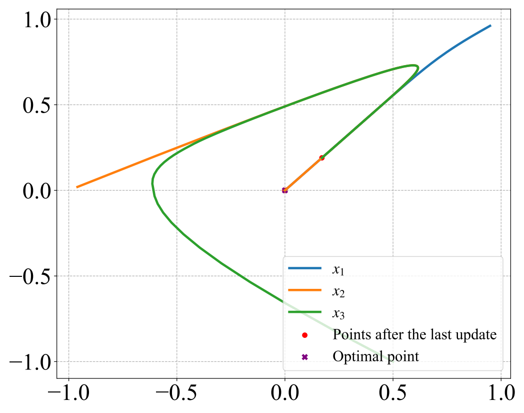


(a) $T_2 = 10, T_3 = 10$

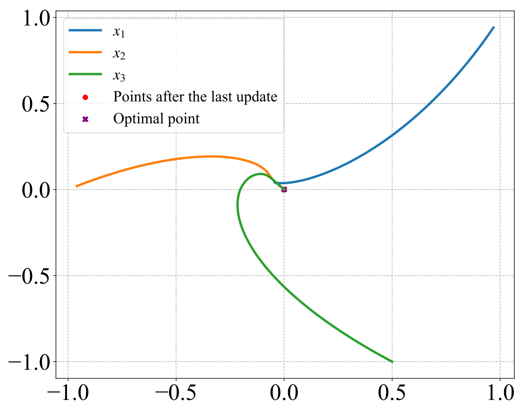


(b) $T_2 = 1, T_3 = 1$

Figure 1: Performance of Algorithm 1 for Problem (8). The objective values linearly decreased.



(a) $T_2 = 10, T_3 = 10$



(b) $T_2 = 1, T_3 = 1$

Figure 2: Trajectories of each variable. $(0, 0)^\top$ is the optimal solution of each level. Our algorithm with few iterations for lower-level problems $T_2 = T_3 = 1$ performed well.

focusing on the number of updates of x_3. We confirmed that the values of f_1, f_2, and f_3 converge to the optimal value 0 at all levels and the gradient method outputs the optimal solution $(0, 0)^\top$ even if we use small iteration numbers $T_2 = 1$ and $T_3 = 1$ for the approximation problem (5).

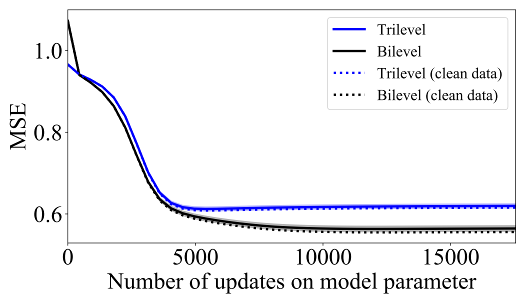
We also made comparison Algorithm 1 and an existing algorithm [23] based on evolutionary strategy by solving (8) using both algorithms. Algorithm 1 outperformed the existing algorithm. For detail, see Supplementary material C.2.

5.2 Application to hyperparameter optimization

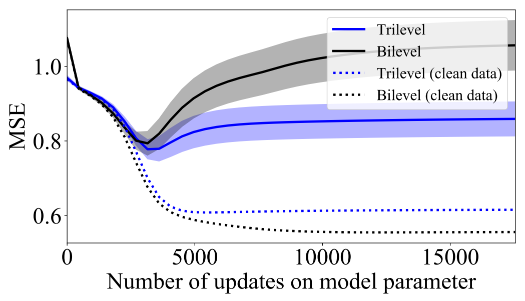
For deriving a machine learning model robust to noise in input data, we formulate a trilevel model by assuming two players: a model learner and an attacker. The model learner decides the hyperparameter λ to minimize the validation error, while the attacker tries to poison training data so as to make the model less accurate. This model is inspired by bilevel hyperparameter optimization [12] and adversarial learning [16, 17] and formulated as follows:

$$\min_{\lambda} \frac{1}{m}\|y_{\text{valid}} - f(X_{\text{valid}};\theta)\|_2^2 \text{ s.t.}$$
$$P \in \operatorname*{argmax}_{P'} \frac{1}{n}\|y_{\text{train}} - f(X_{\text{train}} + P';\theta)\|_2^2 - \frac{c}{nd}\|P'\|_2^2 \text{ s.t.}$$
$$\theta \in \operatorname*{argmin}_{\theta'} \frac{1}{n}\|y_{\text{train}} - f(X_{\text{train}} + P';\theta')\|_2^2 + \exp(\lambda)\frac{\|\theta'\|_{1^*}}{d},$$

where f denotes the output of a three-layer perceptron which has 3 hidden units, θ denotes the parameter of the model f, d denotes the dimension of θ, n denotes the number of the training data

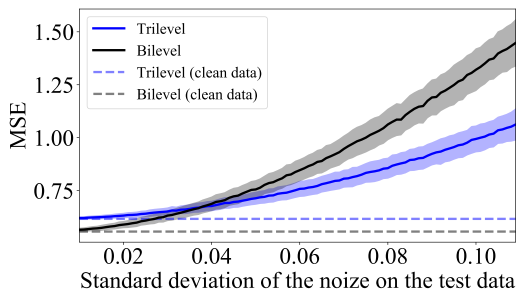


(a) $\sigma = 0.01$

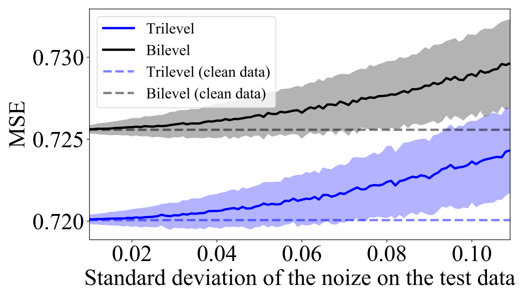


(b) $\sigma = 0.08$

Figure 3: MSE of test data with Gaussian noise with the standard deviation of σ on diabetes dataset.



(a) Diabetes dataset



(b) Red wine quality dataset

Figure 4: MSE of test data with Gaussian noise, using early-stopped parameters for prediction.

X_{train}, m denotes the number of the validation data X_{val}, c denotes the penalty for the noise P, and $\|\cdot\|_{1^*}$ is a smoothed ℓ_1-norm [19, Eq. (18) with $\mu = 0.25$], which is a differentiable approximation of the ℓ_1-norm. We used the hyperbolic tangent function as the activation function for the hidden layer of the multilayer perceptron. Here, we use $\exp(\lambda)$ to express a nonnegative penalty parameter instead of the constraint $\lambda \geq 0$.

To validate the effectiveness of our proposed method, we compared the results by the trilevel model with those of the following bilevel model:

$$\min_{\lambda} \frac{1}{m}\|y_{\text{valid}} - f(X_{\text{valid}};\theta)\|_2^2 \text{ s.t. } \theta \in \operatorname*{argmin}_{\theta'} \frac{1}{n}\|y_{\text{train}} - f(X_{\text{train}};\theta')\|_2^2 + \exp(\lambda)\frac{\|\theta'\|_{1^*}}{d}.$$

This model is equivalent to the trilevel model without the attacker's level.

We used Algorithm 1 to compute the gradient of the objective function in the trilevel and bilevel models with real datasets. For the iterative method in Algorithm 1, we employed the steepest descent method at all levels. We set $T_2 = 30$ and $T_3 = 3$ for the trilevel model and $T_2 = 30$ for the bilevel model. In each dataset, we used the same initialization and step sizes in the updates of λ and θ in trilevel and bilevel models. We compared these methods on the regression tasks with the following datasets: the diabetes dataset [10], the (red and white) wine quality datasets [8], the Boston dataset [14]. For each dataset, we standardized each feature and the objective variable; randomly chose 40 rows for training data $(X_{\text{train}}, y_{\text{train}})$, chose other 100 rows for validation data $(X_{\text{valid}}, y_{\text{valid}})$, and used the rest of the rows for test data.

We show the transition of the mean squared error (MSE) by test data with Gaussian noise in Figure 3. The solid line and colored belt respectively indicate the mean and the standard deviation over 500 times of generation of Gaussian noise. The dashed line indicates the MSE without noise as a baseline. In the results of the diabetes dataset (Figure 3), the trilevel model provided a more robust parameter than the bilevel model, because the MSE of the trilevel model rises less in the large noise setting for test data.

Next, we compared the quality of the resulting model parameters by the trilevel and bilevel models. We set an early-stopping condition on learning parameters: after 1000 times of updates on the model

Table 1: MSE of test data with Gaussian noise with the standard deviation of 0.08, using early-stopped parameters for prediction. The better values are shown in boldface.

	diabetes	Boston	wine (red)	wine (white)
Trilevel	$\mathbf{0.8601 \pm 0.0479}$	$\mathbf{0.4333 \pm 0.0032}$	$\mathbf{0.7223 \pm 0.0019}$	$\mathbf{0.8659 \pm 0.0013}$
Bilevel	1.0573 ± 0.0720	0.4899 ± 0.0033	0.7277 ± 0.0019	0.8750 ± 0.0014

parameter, if one time of update on hyperparameter λ did not improve test error, terminate the iteration and return the parameters at that time. By using the early-stopped parameters, we show the relationship of test error and standard deviation of the noise on the test data in Figure 4 and Table 1. For the diabetes dataset, the growth of MSE of the trilevel model was slower than that of the bilevel model. For wine quality and Boston house-prices datasets, the MSE of the trilevel model was consistently lower than that of the bilevel model. Therefore, the trilevel model provides more robust parameters than the bilevel model in these settings of problems.

5.3 Relationship between (T_2, T_3) and the convergence speed

In the first experiment in Section 5.1, there is no complex relationship between variables at each level, and therefore, the objective function value at one level is not influenced significantly when variables at other levels is changed. In such a problem setting, by setting T_2 and T_3 to small values, we update x_1 many times and the generated sequence $\{x_1^{(t)}\}$ quickly converges to a stationary point. On the other hand, for example, if we set T_3 to a large value, x_3 is well optimized for some fixed x_1 and x_2, and hence, our whole algorithm may need more computation time until convergence. In the second experiment in Section 5.2, the relationship between variables at each level is more complicated than in the first experiment. Setting T_2 or T_3 smaller in such a problem is not necessarily considered to be efficient because the optimization algorithm proceeds without fully approximating the optimality condition of x_i at the ith level.

6 Conclusion

Summary In this paper, we have provided an approximated formulation for a multilevel optimization problem by iterative methods and discussed its asymptotical properties of it. In addition, we have proposed an algorithm for computing the gradient of the objective function of the approximated problem. Using the gradient information, we can solve the approximated problem by the projected gradient method. We have also established the local and global convergence of the projected gradient method.

Limitation and future work Our proposed gradient computation algorithm is fixed-parameter tractable and hence it works efficiently for small n. For large n, however, the exact computation of the gradient is expensive. Development of heuristics for approximately computing the gradient is left for future research. Weakening assumptions in the theoretical contribution is also left for future work. In addition, there is a possibility of another algorithm to solve the approximated problem 5. In this paper, we propose Algorithm 1, which corresponds to forward mode automatic differentiation. On the other hand, in the prior research for bilevel optimization [11], two algorithms were proposed from the perspective of forward mode automatic differentiation and reverse mode automatic differentiation, respectively. Therefore, there is a possibility of another algorithm for problem 5 which corresponds to the reverse mode automatic differentiation, and that is left for future work.

Acknowledgments and Disclosure of Funding

This work was partially supported by JSPS KAKENHI (JP19K15247, JP17H01699, and JP19H04069).

References

[1] G. B. Allende and G. Still. Solving bilevel programs with the KKT-approach. *Mathematical Programming*, 138(1):309–332, 2013.

[2] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind. Automatic differentiation in machine learning: A survey. *Journal of Machine Learning Research*, 18(153):1–43, 2017.

[3] A. Beck. *First-Order Methods in Optimization*. SIAM, 2017.

[4] A. Ben-Tal, A. Goryashko, E. Guslitzer, and A. Nemirovski. Adjustable robust solutions of uncertain linear programs. *Mathematical Programming*, 99(2):351–376, 2004.

[5] K. P. Bennett, J. Hu, X. Ji, G. Kunapuli, and J.-S. Pang. Model selection via bilevel optimization. In *The 2006 IEEE International Joint Conference on Neural Network Proceedings*, pages 1922–1929, 2006.

[6] D. Bertsimas and F. J. C. T. de Ruiter. Duality in two-stage adaptive linear optimization: Faster computation and stronger bounds. *INFORMS Journal on Computing*, 28(3):500–511, 2016.

[7] A. Billionnet, M.-C. Costa, and P.-L. Poirion. 2-stage robust MILP with continuous recourse variables. *Discrete Applied Mathematics*, 170:21–32, 2014.

[8] P. Cortez, A. Cerdeira, F. Almeida, T. Matos, and J. Reis. Modeling wine preferences by data mining from physicochemical properties. *Decision Support Systems*, 47:547–553, 2009.

[9] A. L. Dontchev and T. Zolezzi. *Well-Posed Optimization Problems*. Springer-Verlag Berlin Heidelberg, 1993.

[10] D. Dua and C. Graff. UCI machine learning repository, 2017. `http://archive.ics.uci.edu/ml`.

[11] L. Franceschi, M. Donini, P. Frasconi, and M. Pontil. Forward and reverse gradient-based hyperparameter optimization. In D. Precup and Y. W. Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1165–1173, 2017.

[12] L. Franceschi, P. Frasconi, S. Salzo, R. Grazzi, and M. Pontil. Bilevel programming for hyperparameter optimization and meta-learning. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1568–1577, 2018.

[13] L. Franceschi, M. Niepert, M. Pontil, and X. He. Learning discrete structures for graph neural networks. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 1972–1982, 2019.

[14] D. Harrison and D. L. Rubinfeld. Hedonic prices and the demand for clean air. *Journal of Environmental Economics and Management*, 5:81–102, 1978.

[15] W. W. Hogan. Point-to-set maps in mathematical programming. *SIAM Review*, 15:591–603, 1973.

[16] M. Jagielski, A. Oprea, B. Biggio, C. Liu, C. Nita-Rotaru, and B. Li. Manipulating machine learning: Poisoning attacks and countermeasures for regression learning. In *2018 IEEE Symposium on Security and Privacy*, pages 19–35, 2018.

[17] S. Liu, S. Lu, X. Chen, Y. Feng, K. Xu, A. Al-Dujaili, M. Hong, and U.-M. O'Reilly. Min-max optimization without gradients: Convergence and applications to black-box evasion and poisoning attacks. In H. Daumé III and A. Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 6282–6293, 2020.

[18] J. Lorraine, P. Vicol, and D. Duvenaud. Optimizing millions of hyperparameters by implicit differentiation. In S. Chiappa and R. Calandra, editors, *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108 of *Proceedings of Machine Learning Research*, pages 1540–1552, 2020.

[19] B. Saheya, C. T. Nguyen, and J.-S. Chen. Neural network based on systematically generated smoothing functions for absolute value equation. *Journal of Applied Mathematics and Computing*, 61:533–558, 2019.

[20] A. Shaban, C.-A. Cheng, N. Hatch, and B. Boots. Truncated back-propagation for bilevel optimization. In K. Chaudhuri and M. Sugiyama, editors, *Proceedings of the Twenty-Second International Conference on Artificial Intelligence and Statistics*, volume 89 of *Proceedings of Machine Learning Research*, pages 1723–1732, 2019.

[21] A. Shafiei, V. Kungurtsev, and J. Marecek. Trilevel and multilevel optimization using monotone operator theory, 2021. arXiv:2105.09407v1.

[22] 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, 2018.

[23] S. L. Tilahun, S. M. Kassa, and H. C. Ong. A new algorithm for multilevel optimization problems using evolutionary strategy, inspired by natural adaptation. In *PRICAI 2012: Trends in Artificial Intelligence*, pages 577–588, 2012.

[24] L. N. Vicente and P. H. Calamai. Bilevel and multilevel programming: A bibliography review. *Journal of Global Optimization*, 5:291–306, 1994.

[25] X. Wu and A. J. Conejo. An efficient tri-level optimization model for electric grid defense planning. *IEEE Transactions on Power Systems*, 32(4):2984–2994, 2017.

[26] İ. Yanikoğlu, B. L. Gorissen, and D. den Hertog. A survey of adjustable robust optimization. *European Journal of Operational Research*, 277(3):799–813, 2019.

Supplementary materials

A Proofs

A.1 Proof of Theorem 2

To prove Theorem 2, we introduce some definitions and a lemma.

Definition 9 ([15]). Let Γ be a point-to-set mapping from a set Θ into a set Ξ, i.e., $\Gamma\colon \Theta \to 2^{\Xi}$.

- Γ is said to be *open* at $\bar{\theta} \in \Theta$ if $\{\theta^k\} \subseteq \Theta$, $\theta^k \to \bar{\theta}$, and $\bar{\xi} \in \Gamma(\bar{\theta})$ imply the existence of an integer m and a sequence $\{\xi^k\} \subseteq \Xi$ such that $\xi^k \in \Gamma(\theta^k)$ for $k \geq m$ and $\xi^k \to \bar{\xi}$.
- Γ is said to be *closed* at $\bar{\theta} \in \Theta$ if $\{\theta^k\} \subseteq \Theta$, $\theta^k \to \bar{\theta}$, $\xi^k \in \Gamma(\theta^k)$, and $\xi^k \to \bar{\xi}$ imply $\bar{\xi} \in \Gamma(\bar{\theta})$.
- Γ is said to be *continuous* at $\bar{\theta} \in \Theta$ if it is both open and closed at $\bar{\theta}$. If Γ is continuous at every $\theta \in \Theta$, it is said to be continuous on Θ.
- Γ is said to be *uniformly compact* near $\bar{\theta} \in \Theta$ if there is a neighborhood N of $\bar{\theta}$ such that the closure of the set $\bigcup_{\theta \in N} \Omega(\theta)$ is compact.

Lemma 10 ([15, Corollary 8.1]). *For an objective function $\varphi\colon \mathbb{R}^d \times \Theta \to \mathbb{R}$ and a constraint mapping $\Omega\colon \Theta \to 2^{\mathbb{R}^d}$, consider the following parametric optimization problem with parameter $\theta \in \Theta \subseteq \mathbb{R}^p$:*

$$\begin{aligned} &\min_{\xi} \varphi(\xi, \theta) \\ &\text{s.t. } \xi \in \Omega(\theta). \end{aligned}$$

For this problem, define the optimal set mapping $\Xi^*\colon \Theta \to 2^{\mathbb{R}^d}$ as

$$\Xi^*(\theta) \coloneqq \{\xi \in \Omega(\theta)\colon \varphi(\xi, \theta) = \inf_{\xi' \in \Omega(\theta)} \varphi(\xi', \theta)\}.$$

Suppose that Ω is continuous at $\bar{\theta} \in \Theta$, φ is continuous on $\Omega(\bar{\theta}) \times \{\bar{\theta}\}$, Ξ^* is nonempty and uniformly compact near $\bar{\theta}$, and $\Xi^*(\bar{\theta})$ is a singleton. Then, Ξ^* is continuous at $\bar{\theta}$.

Proof of Theorem 2. Since S_1 is compact from Assumption 1-(i), it follows from the Weierstrass theorem that a sufficient condition for the existence of minimizers is that F_1 is continuous. Since F_1 is a composition of function f_1, which is continuous from Assumption 1-(ii), and functions $x_2^*, \dots, x_n^*$, we inductively show the continuity of $x_2^*, \dots, x_n^*$ in what follows.

For any $i = 2, \dots, n$, we derive the continuity of x_i^* assuming the continuity of $x_{i+1}^*, \dots, x_n^*$. Since f_i is continuous from Assumption 1-(ii) and so are $x_{i+1}^*, \dots, x_n^*$ from the induction hypothesis, F_i is continuous. In addition, we define $X_i^*\colon S_1 \times \mathbb{R}^{d_2} \times \cdots \times \mathbb{R}^{d_{i-1}} \to 2^{\mathbb{R}^{d_i}}$ as

$$X_i^*(x_1, \dots, x_{i-1}) \coloneqq \left\{ x_i \in \mathbb{R}^{d_i}\colon F_i(x_1, \dots, x_{i-1}, x_i) = \inf_{x_i' \in \mathbb{R}^{d_n}} F_i(x_1, \dots, x_{i-1}, x_i') \right\}.$$

For any $(\bar{x}_1, \dots, \bar{x}_{i-1}) \in S_1 \times \mathbb{R}^{d_2} \times \dots \mathbb{R}^{d_{i-1}}$, X_i^* is nonempty near $(\bar{x}_1, \dots, \bar{x}_{i-1})$ and $X_i^*(\bar{x}_1, \dots, \bar{x}_{i-1})$ is a singleton from Assumption 1-(iii); and X_i^* is uniformly bounded near $(\bar{x}_1, \dots, \bar{x}_{i-1})$ from Assumption 1-(iv). Therefore, from Lemma 10, X_i^* is continuous at $(\bar{x}_1, \dots, \bar{x}_{i-1})$. From the arbitrariness of $(\bar{x}_1, \dots, \bar{x}_{i-1})$, X_i^* is continuous on $S_1 \times \mathbb{R}^{d_2} \times \dots \mathbb{R}^{d_{i-1}}$. Recalling Assumption 1-(iii) again, we have $X_i^*(x_1, \dots, x_{i-1}) = \{x_i^*(x_1, \dots, x_{i-1})\}$. The continuity (in the sense of point-to-set mappings) and single-valuedness of X_i^* means the continuity (in the sense of functions) of x_i^*. □

A.2 Proof of Theorem 4

To prove Theorem 4, we use the following lemma.

Lemma 11 ([12, Theorem A.1]). *Let φ_T and φ be continuous*[2] *functions defined on a compact set Ω. Suppose that φ_T converges uniformly to φ on Ω as $T \to \infty$. Then,*

[2]In the original statement of [12, Theorem A.1], the lower-semicontinuity is assumed, but in the proof, the continuity is used. [12, Theorem A.1] would be rewriting of [9, Theorem 14]. In the statement of [9, Theorem 14], the continuity is assumed.

(a) $\inf_{\xi\in\Omega}\varphi_T(\xi)\to\inf_{\xi\in\Omega}\varphi(\xi)$ *as* $T\to\infty$*;*

(b) $\operatorname{argmin}_{\xi\in\Omega}\varphi_T(\xi)\to\operatorname{argmin}_{\xi\in\Omega}\varphi(\xi)$ *as* $T\to\infty$*, meaning that, for every* $\{\xi_T\}_{T\in\mathbb{N}}$ *such that* $\xi_T\in\operatorname{argmin}_{\xi\in\Omega}\varphi_T(\xi)$*, we have that:*

- $\{\xi_T\}$ *admits a convergent subsequence;*
- *for every subsequence* $\{\xi_{T_k}\}$ *such that* $\xi_{T_k}\to\bar{\xi}$ *as* $k\to\infty$*, we have* $\bar{\xi}\in\operatorname{argmin}_{\xi\in\Omega}\varphi(\xi)$.

Proof of Theorem 4. From Assumption 3-(i), there exists $\nu>0$ such that for every $T\in\mathbb{N}$ and every $x_1\in S_1$,

$$\begin{aligned}|\tilde{F}_1(x_1)-F_1(x_1)| &= |f_1(x_1,x_2^{(T)}(x_1),\ldots,x_n^{(T)}(x_1))-f_1(x_1,x_2^*(x_1),\ldots,x_n^*(x_1))|\\ &\le \nu\sum_{i=2}^{n}\|x_i^{(T)}(x_1)-x_i^*(x_1)\|.\end{aligned}$$

It follows from Assumption 3-(ii) that $\tilde{F}_1$, which depends on T, converges to F_1 uniformly on S_1 as $T\to\infty$. Then, the statement follows from Lemma 11. Note that the continuity of F_1 has already shown in Theorem 2. □

A.3 Proof of Theorem 5

Proof. By using the chain rule, we obtain

$$\nabla_{x_1}\tilde{F}_1(x_1)=\nabla_{x_1}f_1+\sum_{i=2}^{n}\nabla_{x_1}x_i^{(T_i)}(x_1,x_2^{(T_2)}(x_1),\ldots,x_{i-1}^{(T_{i-1})}(x_{k-2}^{(T_{k-2})}(\cdots(x_2^{(T_2)}(x_1)))))\nabla_{x_i}f_1,$$

where we denoted $\nabla_{x_i}f_1=\nabla_{x_i}f_1(x_1,x_2^{(T_2)},\ldots,x_n^{(T_n)})$ for $i=2,\ldots,n$ and $\nabla_{x_1}f_1=\nabla_{x_1}f_1(x_1,x_2^{(T_2)},\ldots,x_n^{(T_n)})$. Hereinafter, for $i=2,\ldots,n$ and $t=1,\ldots,T_i$, we denote $Y_i^{(t)}=\nabla_{x_1}x_i^{(t)}(x_1,x_2^{(T_2)}(x_1),\ldots,x_{i-1}^{(T_{i-1})}(x_{k-2}^{(T_{k-2})}(\cdots(x_2^{(T_2)}(x_1)))))$, and show $Y_i^{(T_i)}=Z_i$ for all $i=2,\ldots,n$. In what follows, we denote $\bar{B}_i^{(t)}=\sum_{j=2}^{i-1}Z_jC_{ij}^{(t)}+B_i^{(t)}$.

For $i=3,\ldots,n$ and $t=1,\ldots,T_i$, by applying the chain rule to the update formula of x_i, i.e., $x_i^{(t)}=\Phi_i^{(t)}(x_1,x_2^{(T_2)},\ldots,x_{i-1}^{(T_{i-1})},x_i^{(t-1)})$, we obtain

$$\begin{aligned}Y_i^{(t)} &= Y_i^{(t-1)}\nabla_{x_i}\Phi_i^{(t)}(x_1,x_2^{(T_2)},\ldots,x_{i-1}^{(T_{i-1})},x_i^{(t-1)})\\ &\quad+\sum_{j=2}^{i-1}Y_j^{(T_j)}\nabla_{x_j}\Phi_i^{(t)}(x_1,x_2^{(T_2)},\ldots,x_{i-1}^{(T_{i-1})},x_i^{(t-1)})\\ &\quad+\nabla_{x_1}\Phi_i^{(t)}(x_1,x_2^{(T_2)},\ldots,x_{i-1}^{(T_{i-1})},x_i^{(t-1)})\\ &= Y_i^{(t-1)}A_i^{(t)}+\sum_{j=2}^{i-1}Y_j^{(T_j)}C_{ij}^{(t)}+B_i^{(t)}.\end{aligned}\tag{9}$$

For $i=2$ and $t=1,\ldots,T_2$, note that the arguments of $\Phi_2^{(t)}$ are x_1 and $x_2^{(t-1)}$. For $t=1,\ldots,T_2$, by applying the chain rule, we obtain the following:

$$\begin{aligned}Y_2^{(t)} &= Y_2^{(t-1)}\nabla_{x_2}\Phi_2^{(t)}(x_1,x_2^{(t-1)})+\nabla_{x_1}\Phi_2^{(t)}(x_1,x_2^{(t-1)})\\ &= Y_2^{(t-1)}A_2^{(t)}+B_2^{(t)}\\ &= Y_2^{(t-1)}A_2^{(t)}+\bar{B}_2^{(t)}.\end{aligned}\tag{10}$$

Note that, $Y_i^{(0)}=\nabla_{x_1}x_i^{(0)}=O$ for $i=2,\ldots,n$ since the initial point $x_i^{(0)}$ is independent to the other variables. Using (10) repeatedly, we obtain

$$\begin{aligned}Y_2^{(T_2)} &= Y_2^{(T_2-1)}A_2^{(T_2)}+\bar{B}_2^{(T_2)}\\ &= (Y_2^{(T_2-2)}A_2^{(T_2-1)}+\bar{B}_2^{(T_2-1)})A_2^{(T_2)}+\bar{B}_2^{(T_2)}\\ &= ((Y_2^{(T_2-3)}A_2^{(T_2-2)}+\bar{B}_2^{(T_2-2)})A_2^{(T_2-1)}+\bar{B}_2^{(T_2-1)})A_2^{(T_2)}+\bar{B}_2^{(T_2)}\end{aligned}$$

$$
\begin{aligned}
&= \dots \\
&= (((\cdots(\bar{B}_2^{(1)}A_2^{(2)} + \bar{B}_2^{(2)})\cdots)A_2^{(T_2-2)} + \bar{B}_2^{(T_2-2)})A_2^{(T_2-1)} + \bar{B}_2^{(T_2-1)})A_2^{(T_2)} + \bar{B}_2^{(T_2)} \\
&= \sum_{t=1}^{T_2} \bar{B}_2^{(t)} \prod_{s=t+1}^{T_2} A_2^{(s)} \\
&= Z_2. \qquad (11)
\end{aligned}
$$

Next, for $t = 1, \dots, T_3$, plugging Equation (9) with $i = 3$ into Equation (11), we obtain

$$
Y_3^{(t)} = Y_3^{(t-1)}A_3^{(t)} + Z_2C_{22}^{(t)} + B_3^{(t)} = Y_3^{(t-1)}A_3^{(t)} + \bar{B}_3^{(t)}. \qquad (12)
$$

Repeatedly using Equation (12) similarly to the derivation of Equation (11), we obtain

$$
Y_3^{(T_3)} = \sum_{t=1}^{T_3} \bar{B}_3^{(t)} \prod_{s=t+1}^{T_3} A_3^{(s)} = Z_3.
$$

For $i \geq 4$, we can use a similar argument incrementing i: When we evaluate $Y_i^{(T_i)}$, since we already have $Y_j^{(T_j)} = Z_j$ for $l = 2, \dots, k-1$, we can derive the followings:

$$
\begin{aligned}
Y_i^{(t)} &= Y_i^{(t-1)}A_i^{(t)} + \sum_{j=2}^{i-1} Y_j^{(T_j)}C_{ij}^{(t)} + B_i^{(t)} \\
&= Y_i^{(t-1)}A_i^{(t)} + \sum_{j=2}^{i-1} Z_jC_{ij}^{(t)} + B_i^{(t)} \\
&= Y_i^{(t-1)}A_i^{(t)} + \bar{B}_i^{(t)}. \qquad (13)
\end{aligned}
$$

Repeatedly using Equation (13) similarly to the derivation of Equation (11), we obtain $Y_i^{(T_i)} = Z_i$ for $i = 2, \dots, n$. ☐

A.4 Proof of Theorem 6

To prove Theorem 6, we introduce the following lemma.

Lemma 12 (Complexity of Jacobian-vector products [2, 11])**.** *Let $\varphi\colon \mathbb{R}^d \to \mathbb{R}^m$ be a differentiable function and suppose it can be evaluated in time $c(d, m)$ and requires space $s(d, m)$. Denote let $J_\varphi \in \mathbb{R}^{m \times d}$ be the Jacobian matrix of φ. Then, for any vector $\xi \in \mathbb{R}^d$, the product $J_\varphi \xi$ can be computed within $\mathrm{O}(c(d, m))$ time complexity and $\mathrm{O}(s(d, m))$ space complexity.*

On the complexity of Algorithm 1, the following lemma holds. Hereinafter, n times nested $\mathrm{O}(\cdot)$ is denoted by $\mathrm{O}^n(\cdot)$, that is, $\mathrm{O}^n(\cdot) = \mathrm{O}(\mathrm{O}(\cdots \mathrm{O}(\cdot)\cdots))$. Note that, for a parameter n, O^n is not equivalent to O while O^c is equivalent to O for a constant c.

Lemma 13 (The complexity of Algorithm 1)**.** *Let the time and space complexity for computing $\nabla_{x_i} \tilde{F}_i(x_i)$ be c_i and s_i, respectively. We assume that those for evaluating $\Phi_i^{(t)}$ are $\mathrm{O}(c_i)$ and $\mathrm{O}(s_i)$. In addition, we assume that the time and space complexity of $\nabla_{x_1} f_1$ and $\nabla_{x_i} f_1$ for $i = 1, \dots, n$ are smaller in the sense of the order than those of `for` loops in lines 2–9 in Algorithm 1. Then, the time and space complexity of Algorithm 1 can be written as*

$$
c_1 = \sum_{i=2}^{n} \mathrm{O}(T_i i d_1 \,\mathrm{O}(c_i)),\; s_1 = \max_{i=2}^{n} \mathrm{O}^2(s_i). \qquad (14)
$$

Proof. For each $i = 2, \dots, n$ and $t = 1, \dots, T_i$, the complexity of lines 5–7 of Algorithm 1 can be evaluated as follows:

- From the assumption, the time and space complexity of the 5th line are $\mathrm{O}(c_i)$ and $\mathrm{O}(s_i)$, respectively.

- The 6th line computes $i-2$ products of Jacobian matrices and a $d_i \times d_1$ matrices. One product is obtained by computing the product of a Jacobian matrix and a vector d_1 times and hence its time and space complexity are $\mathrm{O}(d_1\,\mathrm{O}(c_i))$ and $\mathrm{O}(\mathrm{O}(s_i))$, respectively. Since the 6th line compute $i-2$ products, the overall time and space complexity of the 6th line are $\mathrm{O}(id_1\,\mathrm{O}(c_i))$ and $\mathrm{O}(\mathrm{O}(s_i))$, respectively.
- The 7th line computes the product of a Jacobian matrix and a $d_i \times d_1$ matrix. Its time and space complexity are $\mathrm{O}(d_1\,\mathrm{O}(c_i))$ and $\mathrm{O}(\mathrm{O}(s_i))$, respectively.

Hence, for each $i = 2, \ldots, n$, the time and space complexity of the single run of the `for` loop in lines 2–7 are $\mathrm{O}(T_i i d_1\,\mathrm{O}(c_i))$, $\mathrm{O}(\mathrm{O}(s_i))$ respectively. Therefore, we obtain Equation (14) as the overall time and space complexity of Algorithm 1. □

Proof of Theorem 6. From the assumption, c_i, s_i are the time and space complexity for computing $\nabla_{x_i}\tilde{F}_i(x_i)$. Hence, from Lemma 13, c_i and s_i are written as

$$c_i = \sum_{j=i+1}^{n} \mathrm{O}(T_j j d_j\,\mathrm{O}(c_j)), \quad s_i = \max_{j=i+1}^{n} \mathrm{O}^2(s_j).$$

From the equation above, c_i $(i = 1, \ldots, n-1)$ can be written by using c_n as follows:

$$\begin{aligned}
c_{n-1} &= \mathrm{O}(T_n n d_{n-1}\,\mathrm{O}(c_n)),\\
c_{n-2} &= \mathrm{O}(T_{n-1}(n-1)d_{n-2}\,\mathrm{O}(c_{n-1})) + \mathrm{O}(T_n n d_{n-2}\,\mathrm{O}(c_n))\\
&= \mathrm{O}(T_{n-1}(n-1)d_{n-2}\,\mathrm{O}(\mathrm{O}(T_n n d_{n-1}\,\mathrm{O}(c_n))))\\
&= \mathrm{O}(T_n T_{n-1} n(n-1) d_{n-1} d_{n-2}\,\mathrm{O}(\mathrm{O}(\mathrm{O}(c_n)))),\\
&\vdots\\
c_i &= \mathrm{O}(T_n T_{n-1}\cdots T_{i+1} n(n-1)\cdots(i+1) d_{n-1} d_{n-2}\cdots d_i\,\mathrm{O}^{2(n-i)-1}(c_n)),\\
&\vdots\\
c_1 &= \mathrm{O}(T_n T_{n-1}\cdots T_2 n(n-1)\cdots 2 d_{n-1} d_{n-2}\cdots d_1\,\mathrm{O}^{2n-3}(c_n))\\
&= \mathrm{O}\left(\mathrm{O}^{2n-3}(c_n) n! \prod_{i=1}^{n-1}(T_{i+1} d_i)\right).
\end{aligned}$$

Similarly, s_i $(i = 1, \ldots, n-1)$ can be written by using s_n as follows:

$$\begin{aligned}
s_{n-1} &= \mathrm{O}^2(s_n),\\
s_{n-2} &= \max_{l=n-1}^{n} \mathrm{O}^2(s_l) = \max\{\mathrm{O}^2(s_l), \mathrm{O}^4(s_l)\} = \mathrm{O}^4(s_n),\\
&\vdots\\
s_i &= \mathrm{O}^{2(n-i)}(s_n),\\
&\vdots\\
s_1 &= \mathrm{O}^{2n-2}(s_n).
\end{aligned}$$

Note that, since the gradient of the objective function in the lower level is used to calculate the gradient of the objective function in the upper level, the amount of calculation of the gradient at each level increases from the bottom to the top. Therefore, for some $p, q > 1$, it holds that $\mathrm{O}^{2n-3}(c_n) = \mathrm{O}(p^n c_n)$ and $\mathrm{O}^{2n-2}(s_n) = \mathrm{O}(q^n s_n)$. Hence, we obtain Equation (7) as the overall time and space complexity for computing $\nabla_{x_1}\tilde{F}_1(x_1)$ recursively calling Algorithm 1. □

A.5 Proof of Theorem 7

Proof. In this proof, we inductively show the Lipschitz continuity of $\nabla_{x_i}\tilde{F}_i$ for all $i = 1, \ldots, n$ by using the following notations: When a function φ is Lipschitz continuous on Ω, its Lipschitz constant is denoted by L_φ, i.e., $\|\varphi(\hat{\xi}) - \varphi(\check{\xi})\| \leq L_\varphi \|\hat{\xi} - \check{\xi}\|$ for any $\hat{\xi}, \check{\xi} \in \Omega$. When φ is bounded on Ω, its bound is denoted by M_φ, i.e., $\|\varphi(\xi)\| \leq M_\varphi$ for any $\xi \in \Omega$.

As the base case, we consider the case of $i = n$. Since $\tilde{F}_n = f_n$, the Lipschitz continuity of $\nabla_{x_n} \tilde{F}_n$ follows from the assumption of the Lipschitz continuity of $\nabla_{x_i} f_n$.

In what follows, as the induction step, we derive the Lipschitz continuity of $\nabla_{x_i} \tilde{F}_i$ by assuming the Lipschitz continuity of $\nabla_{x_j} \tilde{F}_j$ for $j = i+1, \dots, n$. To do so, for any $(\hat{x}_1, \dots, \hat{x}_i)$ and $(\check{x}_1, \dots, \check{x}_i)$ in $S_1 \times \mathbb{R}^{d_2} \times \cdots \times \mathbb{R}^{d_i}$, we evaluate $\|\nabla_{x_i} \tilde{F}_i(\hat{x}_1, \dots, \hat{x}_i) - \nabla_{x_i} \tilde{F}_i(\check{x}_1, \dots, \check{x}_i)\|$. In what follows, constants related to $(\hat{x}_1, \dots, \hat{x}_i)$ and $(\check{x}_1, \dots, \check{x}_i)$ are denoted by ones with hat and check sign, e.g., $\hat{Z}_j$ and $\check{Z}_j$. By applying Theorem 5 to $\tilde{F}_i$, we have

$$\begin{aligned}
&\|\nabla_{x_i} \tilde{F}_i(\hat{x}_1, \dots, \hat{x}_i) - \nabla_{x_i} \tilde{F}_i(\check{x}_1, \dots, \check{x}_i)\| \\
&\leq \|\nabla_{x_i} f_i(\hat{x}_1, \dots, \hat{x}_i, \hat{x}_{i+1}^{(T_{i+1})}, \dots, \hat{x}_n^{(T_n)}) - \nabla_{x_i} f_i(\check{x}_1, \dots, \check{x}_i, \check{x}_{i+1}^{(T_{i+1})}, \dots, \check{x}_n^{(T_n)})\| \\
&\quad + \sum_{j=2}^{n} \|\hat{Z}_j \nabla_{x_j} f_i(\hat{x}_1, \dots, \hat{x}_i, \hat{x}_{i+1}^{(T_{i+1})}, \dots, \hat{x}_n^{(T_n)}) - \check{Z}_j \nabla_{x_j} f_i(\check{x}_1, \dots, \check{x}_i, \check{x}_{i+1}^{(T_{i+1})}, \dots, \check{x}_n^{(T_n)})\|
\end{aligned} \tag{15}$$

First, we evaluate the first term of the right-hand-side of Equation (15). From the assumption of the Lipschitz continuity of $\nabla_{x_i} f_i$, we have

$$\begin{aligned}
&\|\nabla_{x_i} f_i(\hat{x}_1, \dots, \hat{x}_i, \hat{x}_{i+1}^{(T_{i+1})}, \dots, \hat{x}_n^{(T_n)}) - \nabla_{x_i} f_i(\check{x}_1, \dots, \check{x}_i, \check{x}_{i+1}^{(T_{i+1})}, \dots, \check{x}_n^{(T_n)})\| \\
&\leq L_{\nabla_{x_i} f_i} \left(\|(\hat{x}_1, \dots, \hat{x}_i) - (\check{x}_1, \dots, \check{x}_i)\| + \sum_{j=i+1}^{n} \|\hat{x}_j^{(T_j)} - \check{x}_j^{(T_j)}\| \right).
\end{aligned}$$

For any $j = i+1, \dots, n$ and $t = 1, \dots, T_j$, recalling the definition of $\hat{x}_j^{(t)}$ and $\check{x}_j^{(t)}$ and invoking the Lipschitz continuity of $\nabla_{x_j} \tilde{F}_j$, we have

$$\begin{aligned}
&\|\hat{x}_j^{(t)} - \check{x}_j^{(t)}\| \\
&= \|\Phi_j^{(t)}(\hat{x}_1, \dots, \hat{x}_i, \hat{x}_{i+1}^{(T_{i+1})}, \dots, \hat{x}_{j-1}^{(T_{j-1})}, \hat{x}_j^{(t-1)}) - \Phi_j^{(t)}(\check{x}_1, \dots, \check{x}_i, \check{x}_{i+1}^{(T_{i+1})}, \dots, \check{x}_{j-1}^{(T_{j-1})}, \check{x}_j^{(t-1)})\| \\
&\leq \|\hat{x}_j^{(t-1)} - \check{x}_j^{(t-1)}\| + \alpha_j^{(t-1)} \left\| \begin{aligned} &\nabla_{x_j} \tilde{F}_j(\hat{x}_1, \dots, \hat{x}_i, \hat{x}_{i+1}^{(T_{i+1})}, \dots, \hat{x}_{j-1}^{(T_{j-1})}, \hat{x}_j^{(t-1)}) \\ &\quad - \nabla_{x_j} \tilde{F}_j(\check{x}_1, \dots, \check{x}_i, \check{x}_{i+1}^{(T_{i+1})}, \dots, \check{x}_{j-1}^{(T_{j-1})}, \check{x}_j^{(t-1)}) \end{aligned} \right\| \\
&\leq \|\hat{x}_j^{(t-1)} - \check{x}_j^{(t-1)}\| + \alpha_j^{(t-1)} L_{\nabla_{x_j} \tilde{F}_j} \left(\begin{aligned} &\|(\hat{x}_1, \dots, \hat{x}_i) - (\check{x}_1, \dots, \check{x}_i)\| \\ &\quad + \sum_{k=i+1}^{j-1} \|\hat{x}_k^{(T_k)} - \check{x}_k^{(T_k)}\| + \|\hat{x}_j^{(t-1)} - \check{x}_j^{(t-1)}\| \end{aligned} \right).
\end{aligned}$$

In addition, for any $j = i+1, \dots, n$, we have

$$\begin{aligned}
&\|\hat{x}_j^{(1)} - \check{x}_j^{(1)}\| \\
&= \|\Phi_j^{(1)}(\hat{x}_1, \dots, \hat{x}_i, \hat{x}_{i+1}^{(T_{i+1})}, \dots, \hat{x}_{j-1}^{(T_{j-1})}, x_j^{(0)}) - \Phi_i^{(1)}(\check{x}_1, \dots, \check{x}_i, \check{x}_{i+1}^{(T_{i+1})}, \dots, \check{x}_{j-1}^{(T_{j-1})}, \check{x}_j^{(0)})\| \\
&\leq \underbrace{\|x_j^{(0)} - \check{x}_j^{(0)}\|}_{=0} + \alpha_j^{(0)} L_{\nabla_{x_i} \tilde{F}_i} \left(\begin{aligned} &\|(\hat{x}_1, \dots, \hat{x}_i) - (\check{x}_1, \dots, \check{x}_i)\| \\ &\quad + \sum_{k=i+1}^{j-1} \|\hat{x}_k^{(T_k)} - \check{x}_k^{(T_k)}\| + \underbrace{\|\hat{x}_j^{(0)} - \check{x}_j^{(0)}\|}_{=0} \end{aligned} \right).
\end{aligned}$$

Using these inequalities repeatedly, we can derive

$$\begin{aligned}
&\|\nabla_{x_i} f_i(\hat{x}_1, \dots, \hat{x}_i, \hat{x}_{i+1}^{(T_{i+1})}, \dots, \hat{x}_n^{(T_n)}) - \nabla_{x_i} f_i(\check{x}_1, \dots, \check{x}_i, \check{x}_{i+1}^{(T_{i+1})}, \dots, \check{x}_n^{(T_n)})\| \\
&\leq N_1 \|(\hat{x}_1, \dots, \hat{x}_i) - (\check{x}_1, \dots, \check{x}_i)\|
\end{aligned}$$

for some positive constant N_1 depending only on $\{\alpha_j^{(t-1)}\}$, $L_{\nabla_{x_i} f_i}$ and $L_{\nabla_{x_i} \tilde{F}_j}$.

Next, we evaluate each summand in the second term of the right-hand-side of Equation (15). To do so, we notice that $\hat{Z}_j$ ($\check{Z}_j$, respectively) is the sum of products of $\hat{A}_j^{(t)}$'s, $\hat{B}_j^{(t)}$'s, and $\hat{C}_{jk}^{(t)}$'s ($\check{A}_j^{(t)}$'s, $\check{B}_j^{(t)}$'s, and $\check{C}_{jk}^{(t)}$'s, respectively). Formally, we can decompose $\hat{Z}_j = \sum_{p=1}^{P} \prod_{q=1}^{Q_p} \hat{X}_{jpq}$, where $\hat{X}_{jpq} \in \bigcup_{t=1}^{T_j} (\{\hat{A}_j^{(t)}, \hat{B}_j^{(t)}\} \cup \bigcup_{k=1}^{j-1} \{\hat{C}_{jk}^{(t)}\})$. Similarly, by appropriately defining $\check{X}_{jpq}$, we can rewrite $\check{Z}_j = \sum_{p=1}^{P} \prod_{q=1}^{Q_p} \check{X}_{jpq}$, where $\check{X}_{jpq} \in \bigcup_{t=1}^{T_j} (\{\check{A}_j^{(t)}, \check{B}_j^{(t)}\} \cup \bigcup_{k=1}^{j-1} \{\check{C}_{jk}^{(t)}\})$ and $\check{X}_{jpq} = \check{A}_j^{(t)}, \check{B}_j^{(t)}, \check{C}_{jk}^{(t)}$ if $\hat{X}_{jpq} = \hat{A}_j^{(t)}, \hat{B}_j^{(t)}, \hat{C}_{jk}^{(t)}$, respectively. Using this notation, we have

$$
\begin{aligned}
&\|\hat{Z}_j \nabla_{x_j} f_i(\hat{x}_1, \ldots, \hat{x}_i, \hat{x}_{i+1}^{(T_{i+1})}, \ldots, \hat{x}_n^{(T_n)}) - \check{Z}_j \nabla_{x_j} f_i(\check{x}_1, \ldots, \check{x}_i, \check{x}_{i+1}^{(T_{i+1})}, \ldots, \check{x}_n^{(T_n)})\| \\
&\leq \sum_{p=1}^{P} \left\| \prod_{q=1}^{Q_p} \hat{X}_{jpq} \nabla_{x_j} f_i(\hat{x}_1, \ldots, \hat{x}_i, \hat{x}_{i+1}^{(T_{i+1})}, \ldots, \hat{x}_n^{(T_n)}) - \prod_{q=1}^{Q_p} \check{X}_{jpq} \nabla_{x_j} f_i(\check{x}_1, \ldots, \check{x}_i, \check{x}_{i+1}^{(T_{i+1})}, \ldots, \check{x}_n^{(T_n)}) \right\| \\
&\leq \sum_{p=1}^{P} \left(\begin{aligned} &\sum_{r=1}^{Q_p} \left\| \begin{aligned} &\prod_{q=1}^{r-1} \check{X}_{jpq} \prod_{q=r}^{Q_p} \hat{X}_{jpq} \nabla_{x_j} f_i(\hat{x}_1, \ldots, \hat{x}_i, \hat{x}_{i+1}^{(T_{i+1})}, \ldots, \hat{x}_n^{(T_n)}) \\ &\qquad - \prod_{q=1}^{r} \check{X}_{jpq} \prod_{q=r+1}^{Q_p} \hat{X}_{jpq} \nabla_{x_j} f_i(\hat{x}_1, \ldots, \hat{x}_i, \hat{x}_{i+1}^{(T_{i+1})}, \ldots, \hat{x}_n^{(T_n)}) \end{aligned} \right\| \\ &+ \left\| \begin{aligned} &\prod_{q=1}^{Q_p} \check{X}_{jpq} \nabla_{x_j} f_i(\hat{x}_1, \ldots, \hat{x}_i, \hat{x}_{i+1}^{(T_{i+1})}, \ldots, \hat{x}_n^{(T_n)}) \\ &\qquad - \prod_{q=1}^{Q_p} \check{X}_{jpq} \nabla_{x_j} f_i(\check{x}_1, \ldots, \check{x}_i, \check{x}_{i+1}^{(T_{i+1})}, \ldots, \check{x}_n^{(T_n)}) \end{aligned} \right\| \end{aligned} \right) \\
&\leq \sum_{p=1}^{P} \left(\begin{aligned} &\sum_{r=1}^{Q_p} \|\hat{X}_{jpr} - \check{X}_{jpr}\| \|\nabla_{x_j} f_i(\hat{x}_1, \ldots, \hat{x}_i, \hat{x}_{i+1}^{(T_{i+1})}, \ldots, \hat{x}_n^{(T_n)})\| \prod_{q=1}^{r-1} \|\check{X}_{jpq}\| \prod_{q=r+1}^{Q_p} \|\hat{X}_{jpq}\| \\ &+ \prod_{q=1}^{Q_p} \|\check{X}_{jpq}\| \left\| \nabla_{x_j} f_i(\hat{x}_1, \ldots, \hat{x}_i, \hat{x}_{i+1}^{(T_{i+1})}, \ldots, \hat{x}_n^{(T_n)}) - \nabla_{x_j} f_i(\check{x}_1, \ldots, \check{x}_i, \check{x}_{i+1}^{(T_{i+1})}, \ldots, \check{x}_n^{(T_n)}) \right\| \end{aligned} \right) \\
&\leq \underbrace{\sum_{p=1}^{P} \left(\sum_{r=1}^{Q_p} L_{X_{jpr}} M_{\nabla_{x_j} f_i} \prod_{q \neq r}^{Q_p} M_{X_{jpq}} + L_{\nabla_{x_j} f_i} \prod_{q=1}^{Q_p} M_{X_{jpq}} \right)}_{=N_2} \|(\hat{x}_1, \ldots, \hat{x}_i) - (\check{x}_1, \ldots, \check{x}_i)\|,
\end{aligned}
$$

where, in the last inequality, we used the Lipschitz continuity and the boundedness of $\nabla_{x_i} \Phi_i^{(t)}$, $\nabla_{x_1} \Phi_i^{(t)}$, and $\nabla_{x_j} \Phi_i^{(t)}$. Therefore, we obtain

$$
\|\nabla_{x_i} \tilde{F}_i(\hat{x}_1, \ldots, \hat{x}_i) - \nabla_{x_i} \tilde{F}_i(\check{x}_1, \ldots, \check{x}_i)\| \leq \left(N_1 + \sum_{j=2}^{n} N_{2j} \right) \|(\hat{x}_1, \ldots, \hat{x}_i) - (\check{x}_1, \ldots, \check{x}_i)\|.
$$

This means the Lipschitz continuity of $\nabla_{x_i} \tilde{F}_i$, and hence, that of $\nabla_{x_1} \tilde{F}_1$ is obtained by induction. $\square$

B An example of applying Algorithm 1

We explain an example of approximated problems to be solved by Algorithm 1 (i.e., problem 5 with $n = 3$) by assuming a simple setting, where we apply the steepest descent method for the lower-level

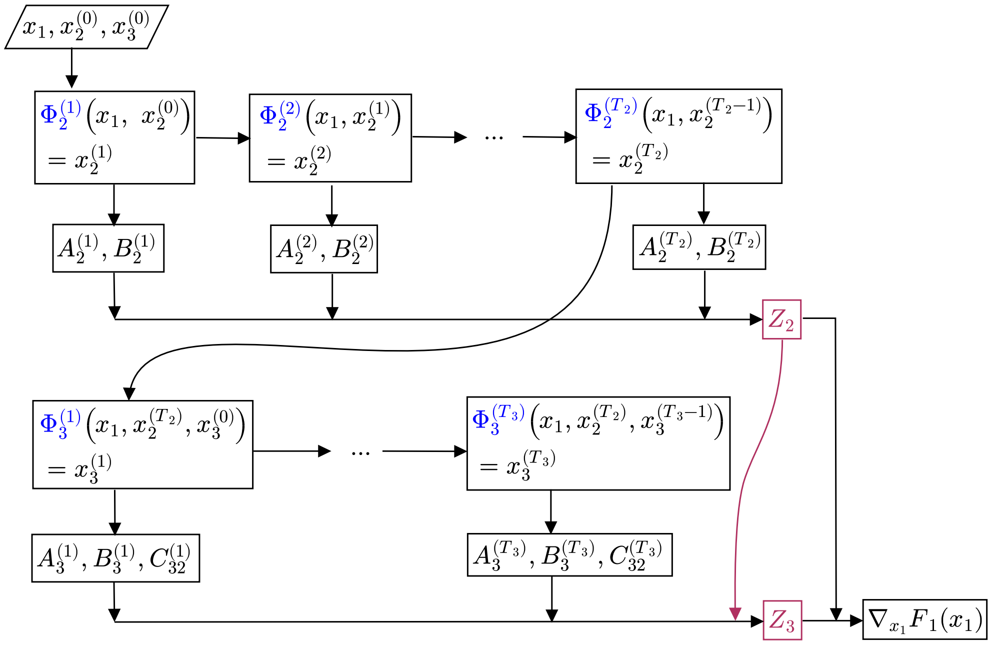

$x_1, x_2^{(0)}, x_3^{(0)}$
$\Phi_2^{(1)}(x_1,\ x_2^{(0)}) = x_2^{(1)}$
$\Phi_2^{(2)}(x_1, x_2^{(1)}) = x_2^{(2)}$
$\Phi_2^{(T_2)}(x_1, x_2^{(T_2-1)}) = x_2^{(T_2)}$
$A_2^{(1)}, B_2^{(1)}$
$A_2^{(2)}, B_2^{(2)}$
$A_2^{(T_2)}, B_2^{(T_2)}$
Z_2
$\Phi_3^{(1)}(x_1, x_2^{(T_2)}, x_3^{(0)}) = x_3^{(1)}$
$\Phi_3^{(T_3)}(x_1, x_2^{(T_2)}, x_3^{(T_3-1)}) = x_3^{(T_3)}$
$A_3^{(1)}, B_3^{(1)}, C_{32}^{(1)}$
$A_3^{(T_3)}, B_3^{(T_3)}, C_{32}^{(T_3)}$
Z_3
$\nabla_{x_1} F_1(x_1)$

Figure 5: Procedure when applying proposed algorithm to trilevel optimization problems.

problems with the same iteration number T and step size α for all levels as follows:

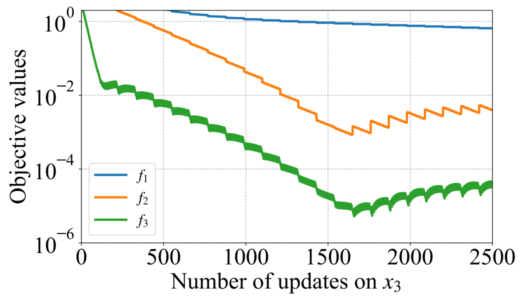
$$\begin{aligned}
\min_{x_1 \in S_1, \{x_2^{(t)}\}, \{x_3^{(t)}\}} \quad & f_1(x_1, x_2^{(T)}, x_3^{(T)}) \\
\text{s.t. } x_2^{(t)} &= x_2^{(t-1)} - \alpha \nabla_{x_2} \tilde{F}_2(x_1, x_2^{(t-1)}), \qquad (t = 1, \ldots, T), \\
x_3^{(t)} &= x_3^{(t-1)} - \alpha \nabla_{x_3} \tilde{F}_3(x_1, x_2^{(T)}, x_3^{(t-1)}) \quad (t = 1, \ldots, T).
\end{aligned} \tag{16}$$

Here, $\nabla_{x_2} \tilde{F}_2(x_1, x_2)$ is the gradient of the objective function of $\min_{x_2 \in \mathbb{R}^{d_2}} \tilde{F}_2(x_1, x_2)$, which is equivalent to the bilevel optimization problem as follows:

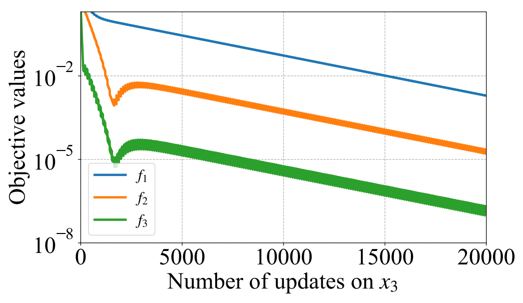
$$\begin{aligned}
\min_{x_2^{(t)}, \{x_3^{(t')}\}} \quad & f_2(x_1, x_2^{(t)}, x_3^{(T)}) \\
\text{s.t. } x_3^{(t')} &= x_3^{(t'-1)} - \alpha \nabla_{x_3} \tilde{F}_3(x_1, x_2^{(t)}, x_3^{(t'-1)}) \quad (t' = 1, \ldots, T).
\end{aligned}$$

In addition, since the third level problem is the lowest level problem, $\tilde{F}_3 = f_3$ holds. We replace $x_2^{(T)}$ in the objective function of (16) by $x_2^{(T-1)} - \alpha \nabla_{x_2} \tilde{F}_2(x_1, x_2^{(T-1)})$ and then replace recursively $x_2^{(t)}$ using $\{x_2^{(t)}\}_{t=0}^{T-1}$. By applying the same procedure for $x_3^{(T)}$, we can reformulate (16) to $\min_{x_1 \in S_1} \tilde{F}_1(x_1)$. Theorem 4 proves that this problem converges to the trilevel optimization problem as $T \to \infty$.

The gradient of $\tilde{F}_1(x_1)$ can be calculated by applying Algorithm 1 to this problem. The explicit formula of $\nabla_{x_1} \tilde{F}_1(x_1)$ is given in Theorem 5 and hence we can compute it by using Algorithm 1. To compute $\nabla_{x_1} \tilde{F}_1(x_1)$ with Algorithm 1, the computation of $\nabla_{x_2} \tilde{F}_2(x_1, x_2)$ is also required. Its explicit formula is also given in Theorem 5 and hence we can compute it by using Algorithm 1. To compute $\nabla_{x_2} \tilde{F}_2(x_1, x_2)$ with Algorithm 1, the computation of $\nabla_{x_3} \tilde{F}_3(x_1, x_2, x_3)$ is also required. This computation is easy (recall $\tilde{F}_3 = f_3$). With these results, we can apply a gradient-based method (e.g., the projected gradient method) to minimize $\tilde{F}_1(x_1)$ using $\nabla \tilde{F}_1$. This procedure of gradient computation is depicted in Figure 5.



(a) Objective values in early iterations



(b) Objective values in later iterations

Figure 6: Performance of Algorithm 1 for Problem (17) ($T_2 = 10$, $T_3 = 10$).

C Complete numerical results

To validate the effectiveness of our proposed method, we conducted numerical experiments on an artificial problem and a hyperparameter optimization problem arising from real data. In our numerical experiments, we implemented all codes with Python 3.9.2 and JAX 0.2.10 for automatic differentiation and executed them on a computer with 12 cores of Intel Core i7-7800X CPU 3.50 GHz, 64 GB RAM, Ubuntu OS 20.04.2 LTS.

C.1 Convergence to optimal solution

We solved the following trilevel optimization problem by using Algorithm 1 to evaluate the performance:

$$\begin{aligned} \min_{x_1\in\mathbb{R}^2} f_1(x_1, x_2^*, x_3^*) = \|x_3^* - x_1\|_2^2 + \|x_1\|_2^2 \text{ s.t.} \\ x_2^* \in \operatorname*{argmin}_{x_2\in\mathbb{R}^2} f_2(x_1, x_2, x_3^*) = \|x_2 - x_1\|_2^2 \text{ s.t.} \\ x_3^* \in \operatorname*{argmin}_{x_3\in\mathbb{R}^2} f_3(x_1, x_2, x_3) = \|x_3 - x_2\|_2^2. \end{aligned} \tag{17}$$

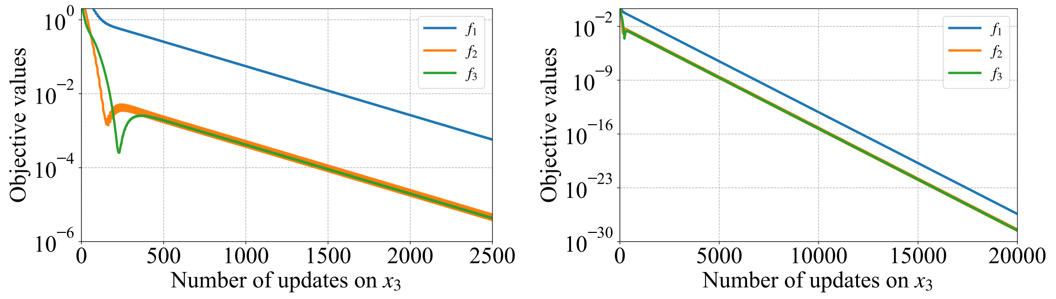
Clearly, the optimal solution for this problem is $x_1 = x_2 = x_3 = (0, 0)^\top$.

We solved (8) with fixed constant step size and initialization but different (T_2, T_3). For the iterative method in Algorithm 1, we employed the steepest descent method at all levels. We show the transition of the value of the objective functions in Figures 6, 7, 8, 9, and 10 and the trajectories of each decision variables in Figure 11. We can confirm that the objective values f_1, f_2, and f_3 converge to the optimal value 0 at all levels and the gradient method outputs the optimal solution $(0, 0)^\top$ even if we use small iteration numbers $T_2 = 1$ and $T_3 = 1$ for the approximation problem (5). In Figures 6, 7, 8, and 9, there is oscillation of the objective values of lower levels, as some decision variable moves away and closer to other decision variables. In Figures 6, 7, and 9, there is a temporary trade-off in the objective values between the levels.

C.2 Comparison between Algorithm 1 and an existing algorithm

We also conducted an experiments to compare the performance of Algorithm 1 with that of an existing method [23], which is based on evolutional strategy. In this experiment, we used the same settings in Section 5.1.

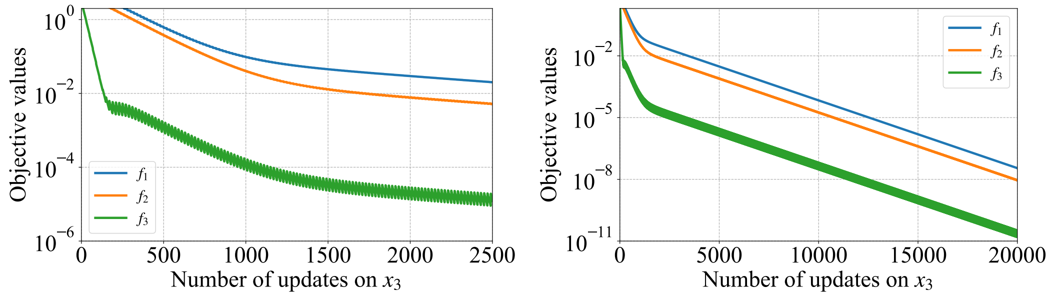
First, we applied the evolutional strategic algorithm [23, Table 1 and Table 2] to the problem 8. To clarify its difference between Algorithm 1, we picked the same initial solution as that of Section 5.1. In [23, `EvolutionaryStrategy` in Table 2], the number of the generations of solution candidates and the number of iterations of perturbations were both set to be 10, the algorithm parameter δ was set to be 10^{-2} and 10^{-4}, and the solution candidates were randomly chosen from the standard normal distribution. We conducted the experiment 100 times for each δ, and we show the transition of the average of the objective functions and the trajectories of each decision variables in Figures 12 and 13.



(a) Objective values in early iterations

(b) Objective values in later iterations

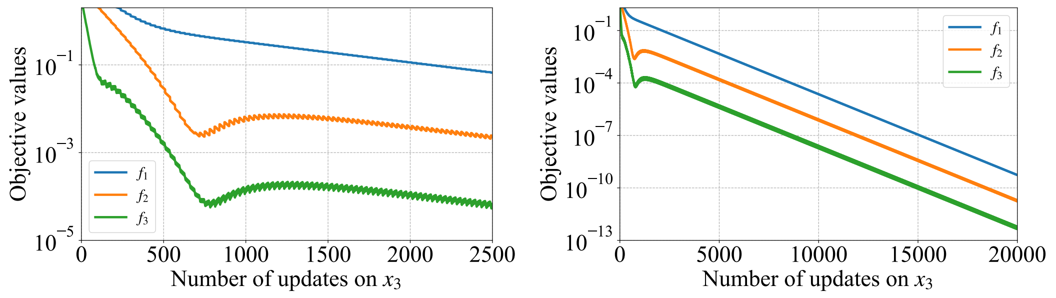
Figure 7: Performance of Algorithm 1 for Problem (17) ($T_2 = 10$, $T_3 = 1$).



(a) Objective values in early iterations

(b) Objective values in later iterations

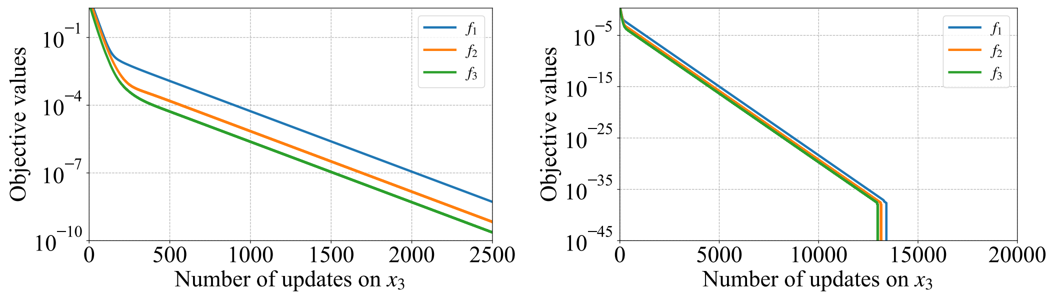
Figure 8: Performance of Algorithm 1 for Problem (17) ($T_2 = 1$, $T_3 = 10$).



(a) Objective values in early iterations

(b) Objective values in later iterations

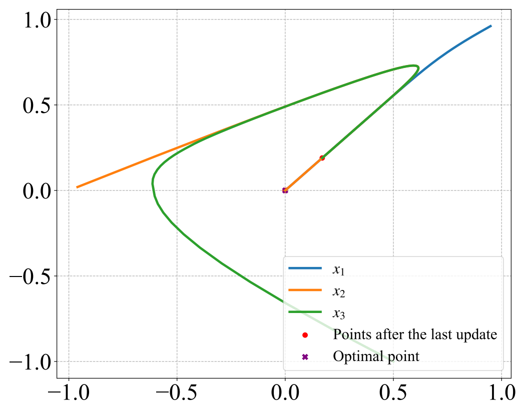
Figure 9: Performance of Algorithm 1 for Problem (17) ($T_2 = 5$, $T_3 = 5$).



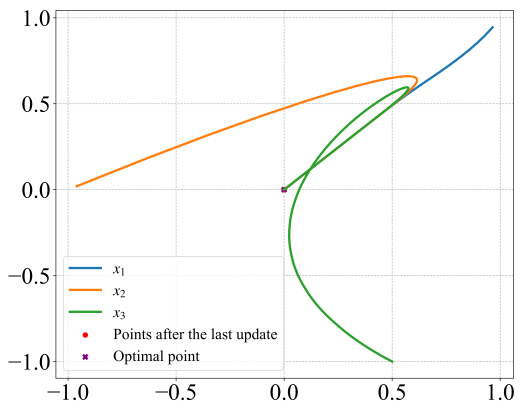
(a) Objective values in early iterations

(b) Objective values in later iterations

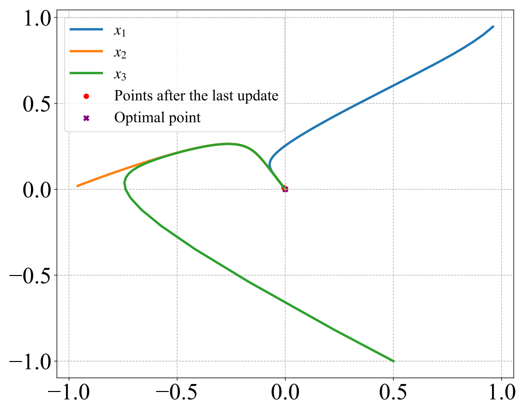
Figure 10: Performance of Algorithm 1 for Problem (17) ($T_2 = 1$, $T_3 = 1$).



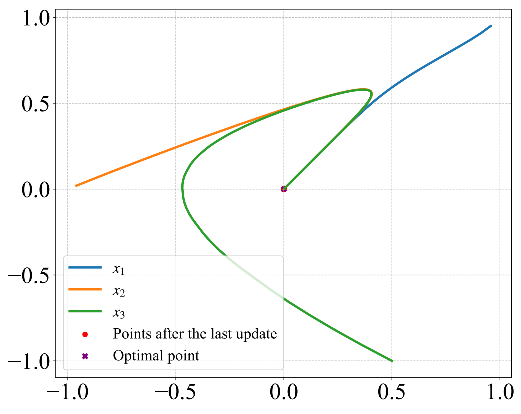
(a) $T_2 = 10, T_3 = 10$



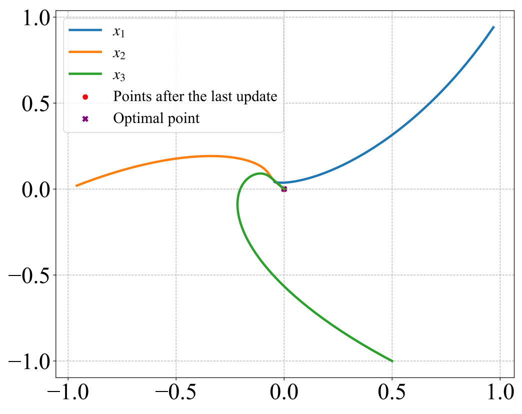
(b) $T_2 = 10, T_3 = 1$



(c) $T_2 = 1, T_3 = 10$

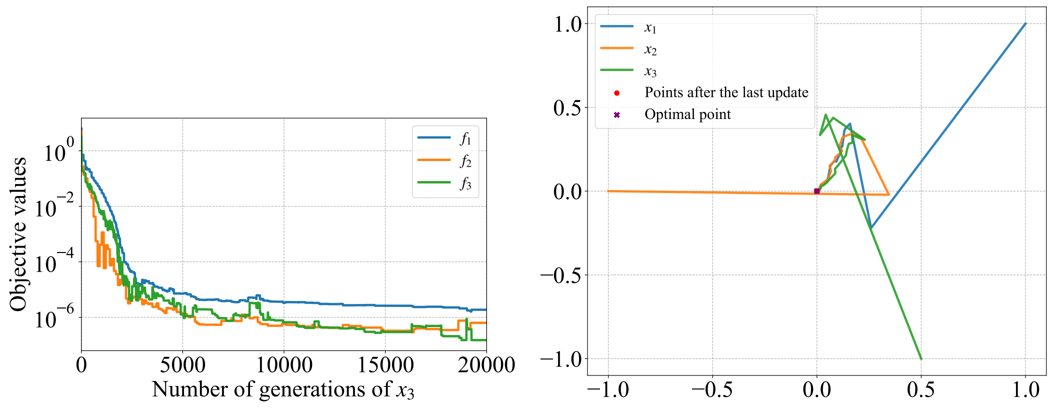


(d) $T_2 = 5, T_3 = 5$



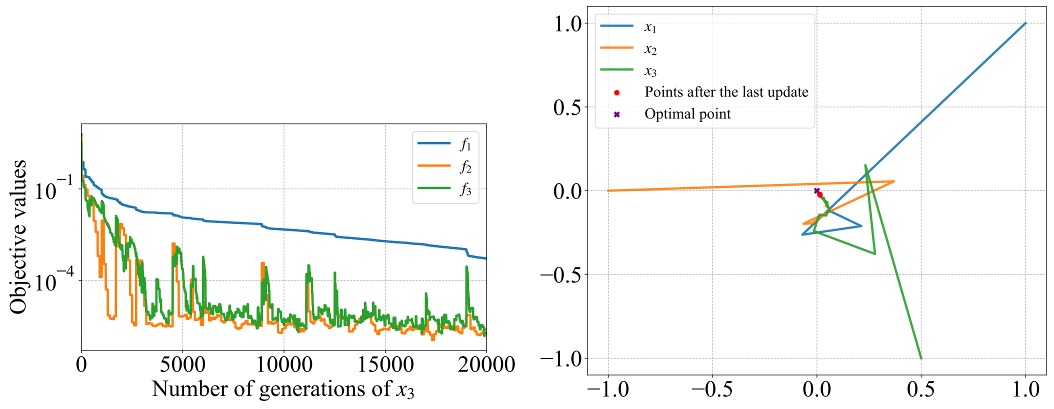
(e) $T_2 = 1, T_3 = 1$

Figure 11: Trajectories of each variables.



(a) Transition of objective values (average of 100 times) (b) Trajectory of each variables (one of 100 times)

Figure 12: Performance of the evolutional strategic algorithm [23] for Problem (17) ($\delta = 10^{-2}$).



(a) Transition of objective values (average of 100 times) (b) Trajectory of each variables (one of 100 times)

Figure 13: Performance of the evolutional strategic algorithm [23] for Problem (17) ($\delta = 10^{-4}$).

Next, we compared the performance of the existing method with that of Algorithm 1. We show the transition of the objective values when using each algorithm in Figure 14 by overlaying Figures 6(b), 10(b), 12(a), and 13(a). In later iterations, we can confirm that the convergence of Algorithm 1 with $(T_2, T_3) = (1, 1)$ is faster than the evolutional strategic algorithm.

C.3 Application to hyperparameter optimization

Next, we conducted experiments on real data to usefulness of our method for a machine learning problem. We consider the problem of how to achieve a model robust to noise in input data and formulate this problem by a trilevel model by a model learner and an attacker. The model learner decides the hyperparameter λ to minimize the validation error, while the attacker tries to poison training data so as to make the model less accurate. This model is formulated as follows:

$$\begin{aligned}
&\min_{\lambda} \frac{1}{m}\|y_{\text{valid}} - f(X_{\text{valid}};\theta)\|_2^2 \text{ s.t.}\\
&\quad P \in \operatorname*{argmax}_{P'} \frac{1}{n}\|y_{\text{train}} - f(X_{\text{train}} + P';\theta)\|_2^2 - \frac{c}{nd}\|P'\|_2^2 \text{ s.t.}\\
&\qquad\qquad \theta \in \operatorname*{argmin}_{\theta'} \frac{1}{n}\|y_{\text{train}} - f(X_{\text{train}} + P';\theta')\|_2^2 + \exp(\lambda)\frac{\|\theta'\|_{1^*}}{d},
\end{aligned}$$

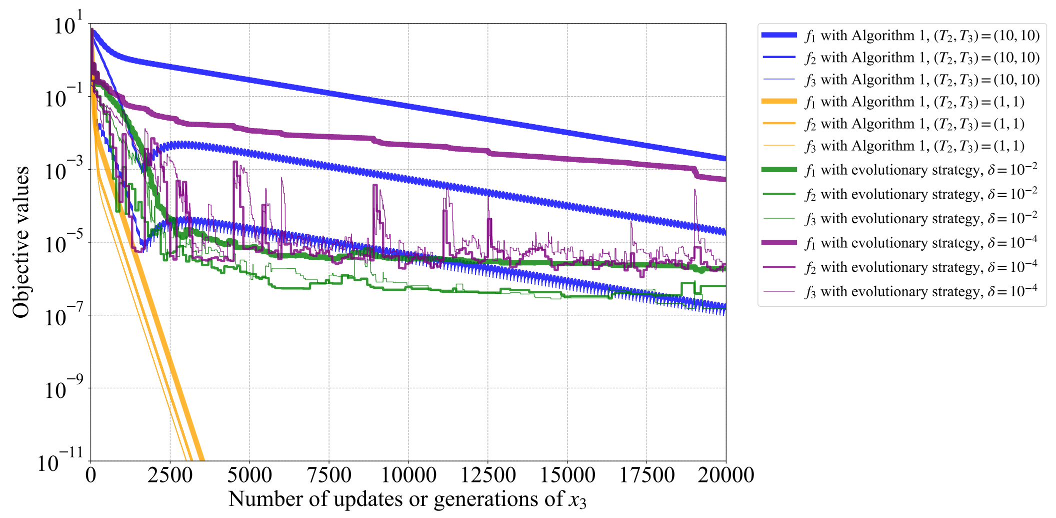


Figure 14: Comparison between the evolutional strategy [23] and Algorithm 1 for Problem (17). Transition of the objective values is plotted. Note that the objective values with the evolutionary strategy are the average of 100 times.

where f denotes the output of a three-layer perceptron which has 3 hidden units, θ denotes the parameter of the model f, d denotes the dimension of θ, n denotes the number of the training data X_{train}, m denotes the number of the validation data X_{val}, c denotes the penalty for the noise P, and $\|\cdot\|_{1^*}$ is a smoothed ℓ_1-norm, which is a differentiable approximation of the ℓ_1-norm. We used the hyperbolic tangent function as the activation function for the hidden layer of the multilayer perceptron. Here, we use $\exp(\lambda)$ to express nonnegative penalty parameter instead of the constraint $\lambda \geq 0$.

To validate the effectiveness of our proposed method, we compared the results by the trilevel model with those of the following bilevel model:

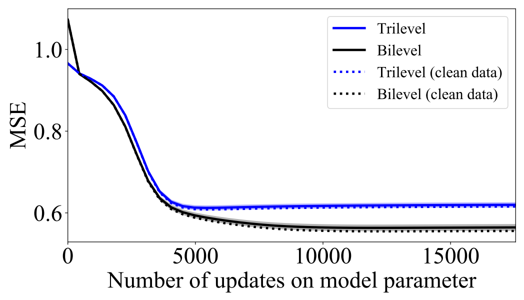
$$\min_{\lambda} \frac{1}{m}\|y_{\text{valid}} - f(X_{\text{valid}};\theta)\|_2^2 \text{ s.t. } \theta \in \operatorname*{argmin}_{\theta'} \frac{1}{n}\|y_{\text{train}} - f(X_{\text{train}};\theta')\|_2^2 + \exp(\lambda)\frac{\|\theta'\|_{1^*}}{d}.$$

This model is equivalent to the trilevel model without the attacker's level.

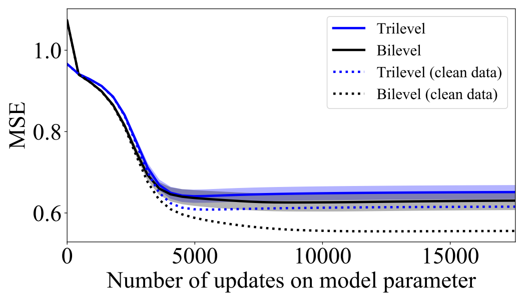
We used Algorithm 1 to compute the gradient of the objective function in the trilevel and bilevel models with real datasets. For the iterative method in Algorithm 1, we employed the steepest descent method at all levels. We set $T_2 = 30$ and $T_3 = 3$ for the trilevel model and $T_2 = 30$ for the bilevel model. In each dataset, we used the same initialization and step sizes in the updates of λ and θ in trilevel and bilevel models. We compared these methods on the regression tasks with the following datasets: the diabetes dataset [10], the (red and white) wine quality datasets [8], the Boston dataset [14]. For each dataset, we standardized each feature and the objective variable; and randomly chose 40 rows as training data $(X_{\text{train}}, y_{\text{train}})$ other chose 100 rows as validation data $(X_{\text{valid}}, y_{\text{valid}})$, and used the rest of the rows as test data.

We show the transition of the MSE of test data with Gaussian noise in Figures 15, 16, 17, and 18. The solid line and colored belt respectively indicate the mean and the standard deviation over 500 times of generation of Gaussian noise. The dashed line indicates the mean squared error (MSE) without noise as a baseline. In the results of the diabetes dataset (Figure 15), the trilevel model provided a more robust parameter than the bilevel model, because the MSE rises less with the large standard deviation of noise on test data.

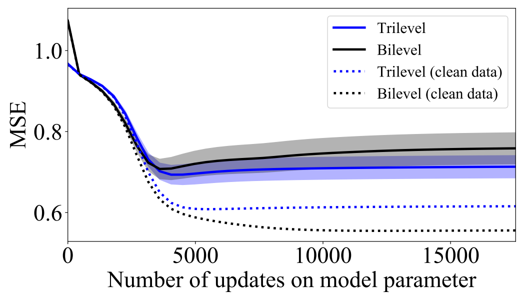
Next, we compared the quality of the resulting model parameters by the trilevel and bilevel models. We set an early-stopping condition on learning parameters: after 1000 times of updates on the model parameter, if one time of update on hyperparameter λ did not improve test error by ϵ, terminate the iteration and return the parameters at that time. By using the early-stopped parameters obtained by this stopping condition, we show the relationship of test error and standard deviation of the noise on the test data in Figure 19 and Table 2. For the diabetes dataset, the growth of MSE of the trilevel model was slower than that of the bilevel model. For wine quality and Boston house-prices datasets,



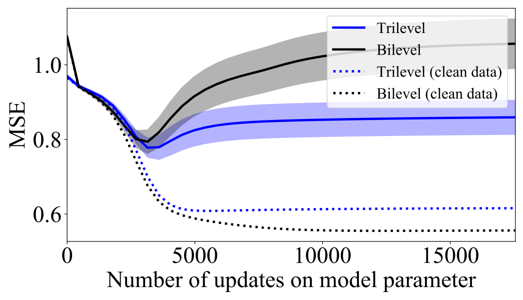
(a) $\sigma = 0.01$



(b) $\sigma = 0.03$

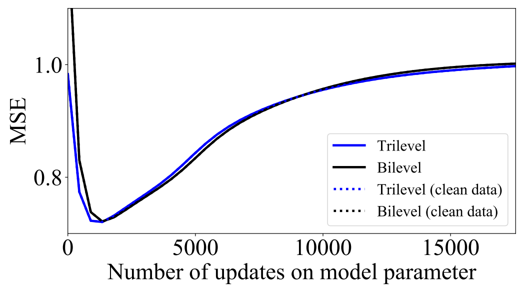


(c) $\sigma = 0.05$

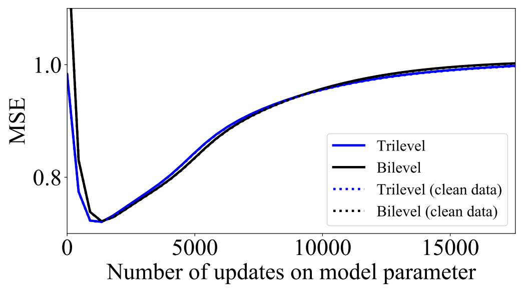


(d) $\sigma = 0.08$

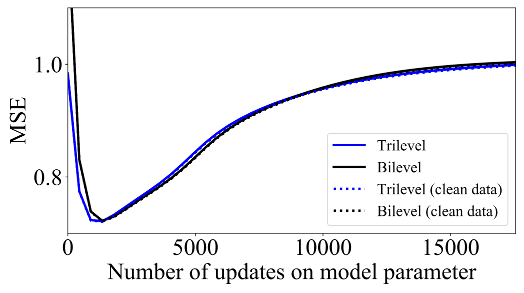
Figure 15: MSE of test data with Gaussian noise with a standard deviation of σ on diabetes dataset.



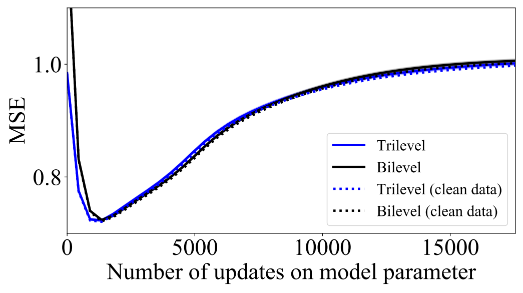
(a) $\sigma = 0.01$



(b) $\sigma = 0.03$

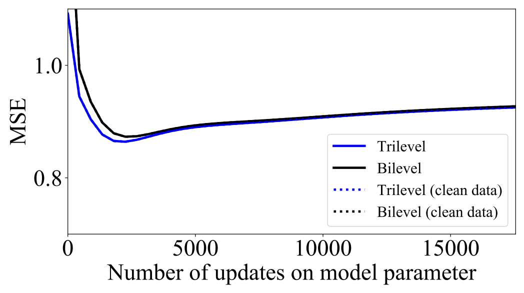


(c) $\sigma = 0.05$

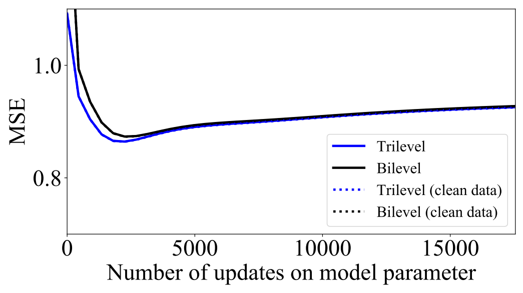


(d) $\sigma = 0.08$

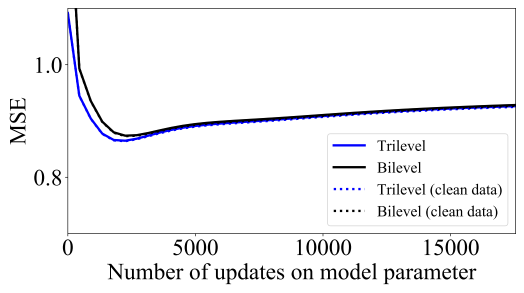
Figure 16: MSE of test data with Gaussian noise with a standard deviation of σ on red wine quality dataset.



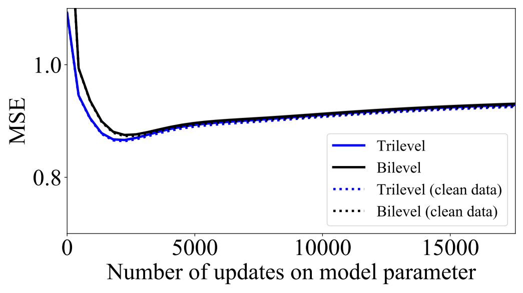
(a) $\sigma = 0.01$



(b) $\sigma = 0.03$

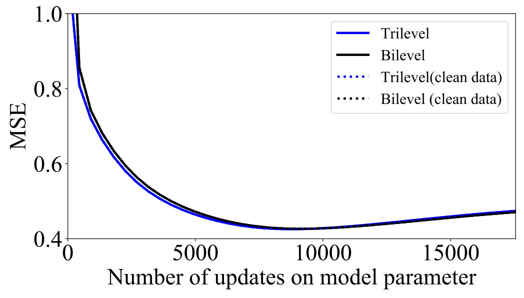


(c) $\sigma = 0.05$

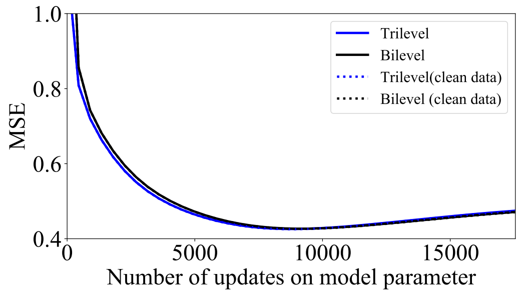


(d) $\sigma = 0.08$

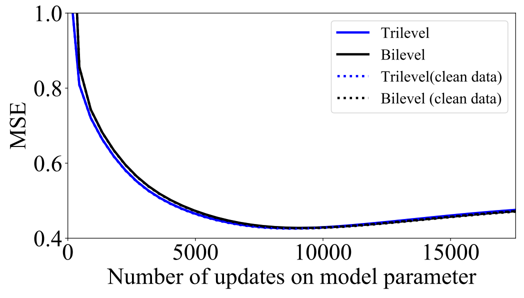
Figure 17: MSE of test data with Gaussian noise with a standard deviation of σ on white wine quality dataset.



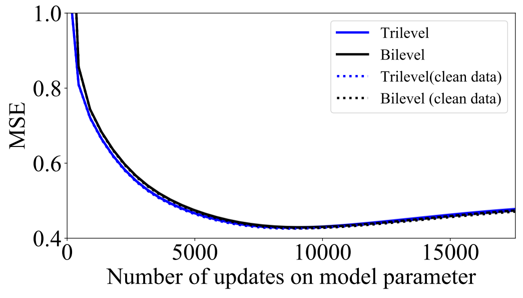
(a) $\sigma = 0.01$



(b) $\sigma = 0.03$

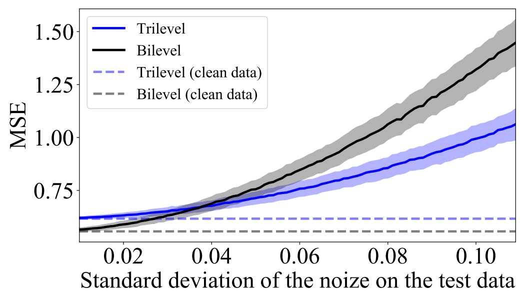


(c) $\sigma = 0.05$

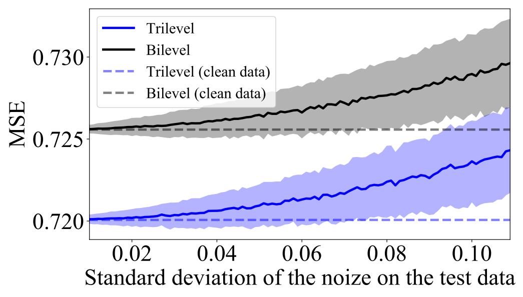


(d) $\sigma = 0.08$

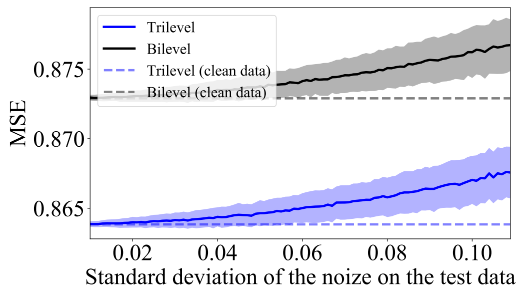
Figure 18: MSE of test data with Gaussian noise with a standard deviation of σ on Boston house-prices dataset.



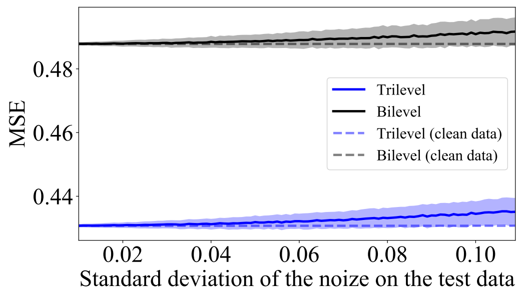
(a) Diabetes dataset



(b) Red wine quality dataset



(c) White wine quality dataset



(d) Boston house-prices dataset

Figure 19: MSE of test data with Gaussian noise, using early-stopped parameters for prediction.

Table 2: MSE of test data with Gaussian noise with standard deviation of 0.08, using early-stopped parameters for prediction. The better values are shown in bold face.

	diabetes	Boston	wine (red)	wine (white)
Trilevel	**0.8601 ± 0.0479**	**0.4333 ± 0.0032**	**0.7223 ± 0.0019**	**0.8659 ± 0.0013**
Bilevel	1.0573 ± 0.0720	0.4899 ± 0.0033	0.7277 ± 0.0019	0.8750 ± 0.0014

the MSE of the trilevel model was consistently lower than that of the bilevel model. Therefore, the trilevel model provides more robust parameters than the bilevel model in these settings of problems.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] See Abstract and Section 1.
   (b) Did you describe the limitations of your work? [Yes] See Section 6.
   (c) Did you discuss any potential negative societal impacts of your work? [No]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] See Sections 3 and 4.
   (b) Did you include complete proofs of all theoretical results? [Yes] See Sections 3 and 4 and Supplementary materials A.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] Our codes are included in the supplementary material.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Section 5.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] See Section 5.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Section 5.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] See Section 5.
   (b) Did you mention the license of the assets? [No] We only used common datasets.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] Our codes are included in the supplementary material.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]