Scalable Kernel Inverse Optimization

Youyuan Long
Delft Center for Systems and Control
Delft University of Technology
The Netherlands
longyouyuan432@gmail.com

Tolga Ok
Delft Center for Systems and Control
Delft University of Technology
The Netherlands
T.Ok@tudelft.nl

Pedro Zattoni Scroccaro
Delft Center for Systems and Control
Delft University of Technology
The Netherlands
P.ZattoniScroccaro@tudelft.nl

Peyman Mohajerin Esfahani
Delft Center for Systems and Control
Delft University of Technology
The Netherlands
P.MohajerinEsfahani@tudelft.nl

Abstract

Inverse Optimization (IO) is a framework for learning the unknown objective function of an expert decision-maker from a past dataset. In this paper, we extend the hypothesis class of IO objective functions to a reproducing kernel Hilbert space (RKHS), thereby enhancing feature representation to an infinite-dimensional space. We demonstrate that a variant of the representer theorem holds for a specific training loss, allowing the reformulation of the problem as a finite-dimensional convex optimization program. To address scalability issues commonly associated with kernel methods, we propose the Sequential Selection Optimization (SSO) algorithm to efficiently train the proposed Kernel Inverse Optimization (KIO) model. Finally, we validate the generalization capabilities of the proposed KIO model and the effectiveness of the SSO algorithm through learning-from-demonstration tasks on the MuJoCo benchmark.

1 Introduction

Inverse Optimization (IO) is distinct from traditional optimization problems, where we typically seek the optimal decision variables by optimizing an objective function over a set of constraints. In contrast, inverse optimization works "in reverse" by inferring the optimization objective given the optimal solution. The inherent assumption in IO is that an agent generates its decision by solving an optimization problem. The assumed optimization problem is called the Forward Optimization Problem (FOP), which is parametric in the exogenous signal $\hat{s}$ with the corresponding optimal solution $\hat{u}$. Therefore, IO aims to deduce the objective function of the FOP from a dataset of exogenous signal and decision pairs, $\{(\hat{s}_i, \hat{u}_i)\}_{i=1}^{N}$. In this work, we assume the constraints are known a priori. Consequently, we can leverage the FOP derived from the expert's dataset by solving it to mimic the expert's behavior when encountering new exogenous signals. IO has garnered widespread attention within several fields, giving rise to numerous studies encompassing both theoretical and applied research. Application domains include vehicle routing [11, 30, 40], transportation system modeling [28, 8], portfolio optimization [24, 38, 22], power systems [9, 15, 31], electric vehicle charging problems [16], network design [14], healthcare problems [5], as well as controller design [2]. For a more detailed discussion on different applications of IO, we refer the readers to the recent survey paper [10].

IO can be categorized into classic IO and data-driven IO. In classic IO, only a single signal-decision pair is considered, where the decision is assumed to be the optimal solution of the FOP (i.e., there is no noise), and different classes of FOPs have been studied, such as linear conic problems [1, 35, 19]. However, in real-world applications, there are usually many observations of signal-decision pairs, and due to the presence of noise, it is usually unreasonable to assume that all observed decisions are optimal w.r.t a single, true data-generating FOP. Additionally, for complex tasks, the chosen FOP may only approximate the task, not allowing for perfect replication of the observed behavior from the expert. These cases are referred to as data-driven IO problems. In such scenarios, a loss function is usually used to compute the discrepancy between observed data and the decision generated by the learned FOP. Examples of loss functions include the *2-norm distance loss* [4], *suboptimality loss* [24], *variational inequality loss* [8], *KKT loss* [20], and *augmented suboptimality loss* [39].

In data-driven IO, the objective function of the FOP is typically non-linear with respect to an exogenous signal $\hat{s}$. Hence, classical methods that learn an FOP based on linear function classes may oversimplify the problem and lead to suboptimal solutions. One effective approach for addressing the expressibility issue in data-driven IO problems is the introduction of kernel methods. These methods have been extensively studied within the context of IO [34, 8] and have shown promising results for scaling IO to address practical problems. The application of kernel methods in IO allows for the exploration of a broader class of optimization problems, thereby enhancing the model's ability to generalize from observed decisions to unseen situations. Specifically, using a kernelized approach facilitates the embedding of decision data into a richer feature space, enabling the deduction of an FOP that not only fits the training data but also exhibits strong generalization capabilities.

Contributions. We list the contributions of this work as follows:

(1) Kernelized IO Formulation: We propose a novel Kernel Inverse Optimization (KIO) model based on suboptimality loss [24]. The proposed approach leverages kernel methods to enable IO models to operate on infinite-dimensional feature spaces, which allows KIO to outperform existing imitation learning (IL) algorithms on complex continuous control tasks in low-data regimes.

(2) Sequential Selection Optimization Algorithm: To address the quadratic computational complexity of the proposed KIO model, we introduce the Sequential Selection Optimization (SSO) algorithm inspired by coordinate descent style updates. This algorithm selectively optimizes components of the decision variable, greatly enhancing efficiency and scalability while provably converging to the same solution of our proposed KIO model.

(3) Open Source Code: To foster reproducibility and further research, we provide an open-source implementation of the proposed KIO model and the SSO algorithm, along with the source code of the experiments in Github[1].

Notation $\mathbb{R}^n_+$ denotes the space of n-dimensional non-negative vectors. The identity square matrix with dimension n is denoted by I_n. For a symmetric matrix Q, the inequality $Q \succeq 0$ (respectively, $Q \succ 0$) means that Q is positive semi-definite (respectively, positive definite). The trace of a matrix Q is denoted as $\text{Tr}(Q)$. Given a vector $x \in \mathbb{R}^n$, we use the shorthand notation $\|x\|_Q^2 := x^\top Qx$. Symmetric block matrices are described by the upper diagonal elements while the lower diagonal elements are replaced by "*". The Frobenius norm of matrix Q is denoted as $\|Q\|_F$. The notation Q_{ij} represents the element in the i-th row and j-th column of the matrix Q. The Euclidean inner product of x and y is denoted as $x^\top y$.

2 Preliminaries

2.1 Inverse Optimization

In general, to solve a data-driven IO problem, we need to design two components: the Forward Optimization Problem (FOP) and the loss function. Specifically, the FOP corresponds to the optimization problem we aim to "fit" to the observed dataset $\hat{\mathcal{D}} = \{(\hat{s}_i, \hat{u}_i)\}_{i=1}^N$, where each input-output pair $(\hat{s}_i, \hat{u}_i) \in \mathcal{S} \times \mathbb{R}^n$. In this paper, we use the "hat" notation (e.g., $\hat{s}$) to denote objects that depend on the dataset. Our goal is to find a parameter vector $\theta \in \Theta$ such that

$$\text{FOP}(\theta, \hat{s}) := \min_{u \in \mathbb{U}(\hat{s})} F_\theta(\hat{s}, u), \tag{1}$$

[1] https://github.com/Longyouyuan/Scalable-Kernel-Inverse-Optimization

replicates the data as closely as possible by minimizing a loss function, akin to classical empirical risk minimization problems. In this work, we focus on lifting the learning problem based on quadratic FOPs. These FOPs include linear constraints and continuous decision variables, as defined by

$$F_\theta(\hat{s}, u) := u^\top \theta_{uu} u + 2\phi(\hat{s})^\top \theta_{su} u \quad \text{and} \quad \mathbb{U}(\hat{s}) := \{u \in \mathbb{R}^n : M(\hat{s})u \le W(\hat{s})\}, \tag{2}$$

where $\theta := (\theta_{uu}, \theta_{su})$, $\theta_{uu} \in \mathbb{R}^{n\times n}$, $\theta_{su} \in \mathbb{R}^{m\times n}$, $M(s) \in \mathbb{R}^{m\times n}$, $W(s) \in \mathbb{R}^m$, and $\phi : \mathcal{S} \to \mathbb{R}^m$ is the feature function that maps $\hat{s}$ to a higher-dimensional feature space to enhance the model's capacity. To simplify notation, we omit the explicit dependence of M and W on s, denoting them as $\hat{M}$ and $\hat{W}$, respectively.

To learn θ, we solve a regularized loss minimization problem using the Suboptimality Loss [24]

$$\min_{\theta \in \Theta} \; k\mathcal{R}(\theta) + \frac{1}{N}\sum_{i=1}^{N} \max_{u_i \in \mathbb{U}(\hat{s}_i)} \left\{F_\theta(\hat{s}_i, \hat{u}_i) - F_\theta(\hat{s}_i, u_i)\right\}, \tag{3}$$

where $\Theta := \{\theta = (\theta_{uu}, \theta_{su}) : \theta_{uu} \succeq I_n\}$, $\mathcal{R}(\theta) := \|\theta_{uu}\|_F^2 + \|\theta_{su}\|_F^2$, and k is a positive regularization parameter. The constraint $\theta_{uu} \succeq I_n$ prevents the trivial solution $\theta_{uu} = \theta_{su} = 0$ and guarantees that the resulting FOP is a convex optimization problem. Moreover, since F_θ is linear in θ, the optimization program (3) is convex w.r.t. θ, and it can be reformulated from the "minimax" form to a single minimization problem. This reformulation is based on dualizing the inner maximization problems of (3) and combining the resulting minimization problems.

Proposition 1 (LMI reformulation [2]). *For the hypothesis function and feasible set in* (2)*, the optimization program* (3) *is equivalent to*

$$\begin{aligned}
\min_{\theta, \lambda_i, \gamma_i} \quad & k\mathcal{R}(\theta) + \frac{1}{N}\sum_{i=1}^{N}\left(F_\theta(\hat{s}_i, \hat{u}_i) + \frac{1}{4}\gamma_i + \hat{W}_i^\top \lambda_i\right) \\
\text{s.t.} \quad & \theta = (\theta_{uu}, \theta_{su}),\ \theta_{uu} \succeq I_n,\ \lambda_i \in \mathbb{R}_+^d,\ \gamma_i \in \mathbb{R} \qquad \forall i \le N \\
& \begin{bmatrix} \theta_{uu} & \hat{M}_i^\top \lambda_i + 2\theta_{su}^\top \phi(\hat{s}_i) \\ * & \gamma_i \end{bmatrix} \succeq 0 \qquad \forall i \le N.
\end{aligned} \tag{4}$$

2.2 The Kernel Method

The kernel method is a powerful technique used in machine learning and statistics that exploits the structure of data embedded in a higher-dimensional space. The kernel method has found numerous applications, including Support Vector Machines [12], Kernel Principal Component Analysis [32], and Kernel Linear Discriminant Analysis [6]. The fundamental idea behind the kernel method is to implicitly map input data into a higher-dimensional space without explicitly computing the transformation, thus enabling algorithms to capture complex patterns and non-linear relationships without heavy computational burden [34]. The kernel method generalizes the hypothesis of the optimization problem to a nonlinear function class based on a *Reproducing Kernel Hilbert Space* (RKHS) $\mathcal{H}$. In this work, we primarily focus on lifting the original parametric optimization problem of the form $\min_{\theta\in\Theta} \ell\left(\{(f_\theta(\hat{s}_i), \hat{s}_i, \hat{u}_i)\}_{i=1}^N, \|\theta\|_F\right)$, where $f_\theta : \mathcal{S} \to \mathbb{R}^n$ such that $f_{\theta_{su}}(s)^\top u := \phi(s)^\top \theta_{su} u$, following Definition (2). The lifted problem is then defined as $\min_{f\in\mathcal{H}} \ell\left(\{(f(\hat{s}_i), \hat{s}_i, \hat{u}_i)\}_{i=1}^N, \|f\|_{\mathcal{H}}\right)$ with f in an RKHS $\mathcal{H}$. By lifting the function class to $\mathcal{H}$, we effectively optimize over *nonlinear* hypotheses.

In a *vector-valued* RKHS $\mathcal{H}$ equipped with the inner product $\langle\cdot,\cdot\rangle_{\mathcal{H}}$, as defined in [23], given a proper kernel function $K : \mathcal{S}\times\mathcal{S} \to \mathbb{R}^{n\times n}$ that is symmetric and positive definite, Moore-Aronszajn's reproducing kernels theory implies that there exists a unique RKHS with the reproducing properties induced by *matrix-valued* K, such that $\forall f \in \mathcal{H} : f(s)^\top u = \langle f, K(\cdot, s)u\rangle_{\mathcal{H}}$ with a linear operator $K(\cdot, s) : \mathbb{R}^n \to \mathcal{H}$. Furthermore, the Riesz representation theorem states that for every $s \in \mathcal{S}$ and $u \in \mathbb{R}^n$, there exists a unique function $K(\cdot, s)u \in \mathcal{H}$ for all $f \in \mathcal{H}$. Recall that the lifted optimization problem is formed over all *nonlinear* hypotheses via $f \in \mathcal{H}$. However, as a result of the reproducing property [33], we can write the lifted optimization problem in the infinite-dimensional inner product space. Hence, the resulting optimization problem has the form

$$\min_{f\in\mathcal{H}} \ell\left(\left\{\left(\langle f, K(\cdot, \hat{s}_i)u\rangle_{\mathcal{H}}, \hat{s}_i, \hat{u}_i\right)\right\}_{i=1}^N, \|f\|_{\mathcal{H}}\right). \tag{5}$$

In what follows, we show that the solution of the optimization problem (5) exists and is finite when the problem is built over a finite dataset $\hat{\mathcal{D}}$ of size N. We leverage the Representer theorem [34], which for an arbitrary loss function, as in (5), states that the solution admits the kernel representation of the form $f^\star(s) = \sum_{i=1}^N K(\hat{s}_i, s)\alpha_i$ with $\alpha_i \in \mathbb{R}^n$. This result effectively suggests that optimizing over an infinite-dimensional RKHS has a sparse solution over the linear hypotheses.

3 Kernel Inverse Optimization

In this section, we extend the Inverse Optimization (IO) model proposed by [2] to incorporate kernel methods. We consider a hypothesis class of the form in Equation (1). However, kernelizing such hypotheses is not straightforward. Instead, we argue that by kernelizing the optimization problem associated with the loss function described in Equation (3), we can obtain a forward optimization problem (FOP) that minimizes the kernelized objective $F(s, u)$.

In Theorem 1, we dualize the problem in (3) and show that the optimal solution for $\theta^\star_{su}$, when plugged into FOP$(\theta^\star_{su}, \theta_{uu}, \hat{s})$, admits an affine function of N coefficients w.r.t $\hat{s}$. The resulting FOP, obtained by Theorem 1, takes a form consistent with the representer theorem when the loss function and kernel are defined appropriately, as discussed later in this section.

Theorem 1 (Kernel reformulation). *The Lagrangian dual of the optimization program* (4) *is*

$$\begin{aligned}
\min_{P,\Lambda_i,\Gamma_i} \quad & \frac{1}{4k}\left\| \left(\sum_{i=1}^N \frac{\hat{u}_i\hat{u}_i^\top}{N} - \Lambda_i\right) - P \right\|_F^2 - \mathrm{Tr}(P) \\
& + \frac{1}{k}\sum_{i=1}^N\sum_{j=1}^N \kappa(\hat{s}_i,\hat{s}_j)\left(\frac{\hat{u}_i}{N} - 2\Gamma_i\right)^\top\left(\frac{\hat{u}_j}{N} - 2\Gamma_j\right) \\
\text{s.t.} \quad & P \succeq 0,\ \Lambda_i \succeq 0,\ \Gamma_i \in \mathbb{R}^n & \forall i \le N \\
& \frac{\hat{s}_i}{N} - 2\hat{M}_i\Gamma_i \ge 0 & \forall i \le N \\
& \begin{bmatrix} \Lambda_i & \Gamma_i \\ * & \frac{1}{4N} \end{bmatrix} \succeq 0 & \forall i \le N,
\end{aligned} \tag{6}$$

where $\kappa(\hat{s}_i,\hat{s}_j) = \phi(\hat{s}_i)^\top\phi(\hat{s}_j)$ *is the* scalar-valued *kernel function. The primal variables* θ_{uu} *and* θ_{su} *can be recovered using*

$$\theta_{uu} = \frac{1}{2k}\left(P - \left(\sum_{i=1}^N \frac{\hat{u}_i\hat{u}_i^\top}{N} - \Lambda_i\right)\right) \quad \text{and} \quad \theta_{su} = \sum_{i=1}^N \phi(\hat{s}_i)\frac{1}{k}\left(2\Gamma_i^\top - \frac{\hat{u}_i^\top}{N}\right). \tag{7}$$

Proof. See Appendix A.1. □

Notice that the complexity of the optimization program (6) does not depend on the dimensionality of the feature vector $\phi(\hat{s}_i)$. Consequently, this allows us to use kernels generated from infinite-dimensional feature spaces, e.g., the Gaussian (a.k.a. radial basis function) kernel $\kappa(\hat{s}_i,\hat{s}_j) = \exp(-\gamma\|\hat{s}_i - \hat{s}_j\|_2^2)$. Program (6) is a convex optimization problem and can be solved using off-the-shelf solvers, such as MOSEK [3]. Once solved, we can recover the optimal primal variables $\theta^\star_{uu}$ and $\theta^\star_{su}$ from the optimal dual variables $\Lambda^\star_i, \Gamma^\star_i$ and $P^\star$ using (7). Notice that the dimensionality of $\theta^\star_{su}$ depends on ϕ, meaning it can be an infinite-dimensional matrix. However, our ultimate goal is to learn an FOP that replicates the behavior observed in the data. By combining (7) with (1), we have that, for the signal $(\hat{s}_{\text{new}}, \hat{M}_{\text{new}}, \hat{W}_{\text{new}})$, the resulting FOP is

$$\min_{\hat{M}_{\text{new}}u \le \hat{W}_{\text{new}}} u^\top \frac{1}{2k}\left(P^\star - \left(\sum_{i=1}^N \frac{\hat{u}_i\hat{u}_i^\top}{N} - \Lambda_i^\star\right)\right)u + \sum_{i=1}^N \kappa(\hat{s}_{\text{new}},\hat{s}_i)\frac{2}{k}\left(2\Gamma_i^\star - \frac{\hat{u}_i}{N}\right)^\top u, \tag{8}$$

which again does not depend on the dimensionality of ϕ, but only on the kernel function κ. However, a key difference between solving the IO problem using the kernel reformulation of Theorem 1 and other classical IO approaches (e.g., [24, 2, 39]) is that the resulting FOP (8) explicitly depends on the entire training dataset $\hat{\mathcal{D}}$. These models are sometimes called *nonparametric models* [25], indicating that the number of parameters of the model (in our case, $P^\star$, $\Lambda_i^\star$, and $\Gamma_i^\star$ for all $i \le N$) depends on the size of the training dataset.

Remark 1 (A potential variant of representer theorem)**.** *In the primal problem* (4)*, a regularized empirical risk loss is optimized over a set of constraints. In the learned objective function of the FOP* (8)*, the term related to the features $\hat{s}_i$ (the linear coefficient of the optimizer u) can be represented as a finite linear combination of kernel products evaluated on the input points in the training set data, i.e., $f^\star(\cdot) = \sum_{i=1}^N \kappa(\hat{s}_i, \cdot)\alpha_i$ with $\alpha \in \mathbb{R}^n$. A similar forward optimization problem can be obtained using a loss function defined in a* vector-valued *RKHS $\mathcal{H}$ with a corresponding kernel function K for functions $f : \mathcal{S} \to \mathbb{R}^n$, as discussed in Section 2.2, via*

$$\ell_{\mathcal{H}}(f, \theta_{uu}, \hat{\mathcal{D}}) = k\|f\|_{\mathcal{H}} + \frac{1}{N}\sum_{i=1}^N \max_{u_i \in \mathbb{U}(\hat{s}_i)} \big\{F(\hat{s}_i, \hat{u}_i; \theta_{uu}, f) - F(\hat{s}_i, u_i; \theta_{uu}, f)\big\},$$

where $F(s, u; \theta_{uu}, f) = u^\top \theta_{uu} u + \langle f, K(\cdot, s)u\rangle_{\mathcal{H}}$ and $K(s, s') \in \mathbb{R}^n \times \mathbb{R}^n$ is set to be a diagonal matrix, with diagonal entries corresponding to the same scalar kernel κ, such that: $K(s, s')_{jj} = \kappa(s, s')$ for $j \in \{1, \ldots, n\}$. Based on the representer theorem, the solution to the optimization problem $\min_{f \in \mathcal{H}} \ell_{\mathcal{H}}(f, \theta_{uu}, \hat{\mathcal{D}})$ admits the form $f^\star(s) = \sum_{i=1}^N K(s, \hat{s}_i)\alpha_i = \sum_{i=1}^N \kappa(s, \hat{s}_i)\alpha_i$. This result indicates that the learned FOP (8) *exhibits characteristics consistent with the representer theorem, implying a potential variant in the context of inverse optimization.*

In the class of cost functions studied in this paper (2), θ_{uu} can be interpreted as a matrix that penalizes the components of the decision vector u. However, in many problems, it is assumed that the expert generating the data equally penalizes each dimension of u, or equivalently, uses $\theta_{uu} = I_n$. This assumption holds, for instance, in the Gymnasium MuJoCo environments [37], where the reward settings for all tasks apply the same penalty to each dimension of the decision vector. Intuitively, this means that the expert trained under such reward settings aims to reduce the magnitude of the decision vector uniformly across all dimensions, rather than favoring any specific dimension. Therefore, assuming $\theta_{uu} = I_n$ as prior knowledge can reduce model complexity and lead to faster training, without degrading model performance.

Corollary 1 (Kernel reformulation for $\theta_{uu} = I_n$)**.** *The Lagrangian dual of the optimization program* (4) *with $\theta_{uu} = I_n$ is*

$$\begin{aligned}
\min_{\Lambda_i, \Gamma_i:\, \forall i \in S} \quad & \frac{1}{k}\sum_{i=1}^N\sum_{j=1}^N \kappa(\hat{s}_i, \hat{s}_j)\left(\frac{\hat{u}_i}{N} - 2\Gamma_i\right)^\top \left(\frac{\hat{u}_j}{N} - 2\Gamma_j\right) + \sum_{i=1}^N \mathrm{Tr}(\Lambda_i) \\
\text{s.t.} \quad & \Lambda_i \succeq 0,\ \Gamma_i \in \mathbb{R}^n && \forall i \in S \\
& \frac{\hat{W}_i}{N} - 2\hat{M}_i\Gamma_i \ge 0 && \forall i \in S \\
& \begin{bmatrix} \Lambda_i & \Gamma_i \\ * & \frac{1}{4N} \end{bmatrix} \succeq 0 && \forall i \in S,
\end{aligned} \tag{9}$$

where $\kappa(\hat{s}_i, \hat{s}_j) = \phi(\hat{s}_i)^\top \phi(\hat{s}_j)$ is the kernel function and $S = \{1, \ldots, N\}$. The primal variable θ_{su} can be recovered using (7)*.*

The proof of Corollary 1 is the same as the proof of Theorem 1 under the assumption that $\theta_{uu} = I_n$, and is therefore omitted here. S represents an index set, where all decision variables whose indices belong to S will be optimized, while decision variables whose indices do not belong to S retain their original values and are treated as constants. In Corollary 1, all variables will be optimized, so S is the set comprising natural numbers from 1 to N. The concept of the index set S is introduced to make Problem (9) compatible with the sub-optimization problems based on the coordinate descent method outlined in Section 4.

In the following section, we will focus on algorithms to solve Problem (9). However, all ideas also apply to the general problem (6).

4 Sequential Selection Optimization

Solving the kernel IO problem (9) involves optimizing a Semidefinite Program (SDP), which can become prohibitively costly if the number of semidefinite constraints and optimization variables grows too large. In our case, the size of the SDP grows quadratically with the size of the training dataset N. For instance, in our experiments, solving (9) with $N = 20000$ using CVXPY [13] requires up to 256GB of memory. Therefore, in this section, we propose a coordinate descent-type algorithm to find an approximate solution to (9) by iteratively optimizing only a subset of the coordinates at each iteration, keeping all other coordinates fixed. We define a pair of variables Λ_i and Γ_i as the i-th coordinate, denoted as $\{\Lambda_i, \Gamma_i\}$. In Problem (9), each coordinate is decoupled in the constraints, which enables the use of the coordinate descent framework here. We call this method *Sequential Selection Optimization* (SSO), and present it in Algorithm 1.

Algorithm 1 Sequential Selection Optimization (SSO)

```
1: Initialize $\{\Lambda_i, \Gamma_i\}_{i=1}^N$
2: for $t = 1, \dots, T$ do
3:     Select a batch of $p$ coordinates $S = \{a_i\}_{i=1}^p$, where $a_i \in \{1, \dots, N\}$
4:     Update $\{\Lambda_{a_i}, \Gamma_{a_i}\}_{i=1}^p$ based on (9) with $S$
5: end for
```

Here, we explain each step of Algorithm 1: (i) Initialization of the optimization variables. In general, the variables are usually initialized randomly or set to 0 or 1. These methods are simple but may not provide a good initial guess. (ii) Selection of a batch of p coordinates. The most straightforward approach to selecting p coordinates is to choose them cyclically. Alternatively, we can select the coordinates at random at each iteration (not necessarily with equal probability). Lastly, we can choose coordinates greedily, selecting the components corresponding to the greatest descent or those with the largest gradient or subgradient at the current iteration [36]. (iii) Solving the KIO subproblem to update the selected coordinates. The mathematical expression of the subproblem is Problem (9) with S, where S is a set containing the indices of the coordinates that need to be updated. Note that the coordinates whose indices $\notin S$ remain fixed. Therefore, the number of quadratic terms in (9) scales with $|S|^2$ rather than N^2.

Next, we propose two heuristics to accelerate the convergence speed of the SSO algorithm: a heuristic method for choosing which coordinates (line 3 of Algorithm 1) to optimize, and a warm-up trick to improve the initialization of the optimization variables (line 1 of Algorithm 1).

4.1 Heuristic for choosing coordinates

At each iteration of Algorithm 1, intuitively, the largest improvement will be made by updating the "least optimal" set of p variables. One way to evaluate their degree of suboptimality is to choose the variables with the most significant violation of the Karush-Kuhn-Tucker (KKT) conditions of the primal version of Program (9) (i.e., (4) with $\theta_{uu} = I_n$), inspired by the Sequential Minimal Optimization (SMO) method from [29].

Proposition 2. *For optimal decision variables of Problem* (9)*, the coordinate* $\{\Lambda_i, \Gamma_i\}$ *that satisfies*

$$\frac{\hat{W}_i}{N} - 2\hat{M}_i\Gamma_i > 0, \tag{10}$$

should also satisfy

$$\mathrm{Tr}\left(\begin{bmatrix}\Lambda_i & \Gamma_i \\ * & \frac{1}{4N}\end{bmatrix}\begin{bmatrix}I_n & 2\theta_{su}^\top\phi(\hat{s}_i) \\ * & \|2\theta_{su}^\top\phi(\hat{s}_i)\|_2^2\end{bmatrix}\right) = 0. \tag{11}$$

Proof. See Appendix A.2. □

Proposition 2 is based on KKT conditions. Based on Condition (11), we can define the KKT violation condition as

$$\texttt{kkt\_violator(i)} := \left|\mathrm{Tr}\left(\begin{bmatrix}\Lambda_i & \Gamma_i \\ * & \frac{1}{4N}\end{bmatrix}\begin{bmatrix}I_n & 2\theta_{su}^\top\phi(\hat{s}_i) \\ * & \|2\theta_{su}^\top\phi(\hat{s}_i)\|_2^2\end{bmatrix}\right)\right|. \tag{12}$$

Using the violation condition (12), we can establish the following heuristic method to construct the set S in Algorithm 1: given the current values of $\{\Lambda_i, \Gamma_i\}_{i=1}^N$ at iteration t, we choose the p coordinates that satisfy Condition (10) with the maximum KKT violation (12). In practice, we additionally select some random coordinates to update at each iteration, ensuring that all coordinates have the chance to be updated, including those that initially do not meet the criteria specified in Condition (10) and would otherwise never be updated. This random selection is inspired by the proven convergence of coordinate descent algorithms with uniformly random and cyclic coordinate selection [26, 7].

4.2 Warm-up trick for improved initialization

Another component of Algorithm 1 that may have a significant practical impact is how the optimization variables $\{\Lambda_i, \Gamma_i\}_{i=1}^N$ are initialized. A poor initial guess (e.g., $\Lambda_i = \Gamma_i = 0$) can lead to slow solver convergence or even result in numerical instability. Here, we propose a simple warm-up trick that leads to a better initialization of the optimization variables. First, we divide the original dataset $\hat{\mathcal{D}}$ into n non-overlapping sub-datasets $\hat{\mathcal{D}}_1, \ldots, \hat{\mathcal{D}}_n$ and solve the n small problems (9) for each of these sub-datasets ($N_i = |\hat{\mathcal{D}}_i|$ and S is the set of indices of all the data in $\hat{\mathcal{D}}_i$). We then concatenate the optimal solutions of these n solved small problems to form an initial guess. Even when the original problem (9) is intractable due to a large training dataset (i.e., large N), each subproblem remains tractable for a small enough batch size N_i, and its solutions are still feasible with respect to (9).

5 Numerical Experiments

5.1 Performance Evaluation

In this evaluation, KIO is implemented in its simplified version (9), incorporating a Gaussian kernel, and tested on continuous control datasets from the D4RL benchmark [17]. The model is trained using the SSO Algorithm 1. In each task, the model's performance is assessed over 100 test episodes, and the score[2] for KIO is the average score across these 100 episodes. The parentheses following KIO scores indicate the amount of data used.

For comparison, four additional agents are selected for this experiment. **IO** is the inverse optimization model without the kernel method, introduced in Proposition 1. To illustrate the effect of the kernel method, both the **KIO** and **IO** models are trained on identical datasets. The scores of two behavior cloning agents, **BC(TD3+BC)** [18] and **BC(CQL)** [21], are taken from two offline reinforcement learning algorithms in which the entire dataset of 1 million samples was used for training. These papers implemented their respective behavior cloning agents using D4RL datasets, serving as baselines to compare against their proposed offline reinforcement learning algorithms. In these studies, **BC(TD3+BC)** and **BC(CQL)** were evaluated over 10 seeds and 3 seeds, respectively. The **Teacher** is the agent responsible for generating the dataset and serves as the target for imitation learning in this experiment.

Table 1: Performance of KIO, IO, two Behavior Cloning (BC) agents, and the Teacher agent on MuJoCo tasks from the D4RL benchmark on the normalized return metric. The numbers in parentheses represent the amount of data used by KIO and IO, and the score for KIO in each task is the average score over 100 episodes.

Task	KIO	IO	BC(TD3+BC)[18]	BC(CQL)[21]	Teacher
Hopper-expert	**109.9** (5k)	31.8	**111.5**	**109.0**	**108.5**
Hopper-medium	**50.2** (5k)	20.6	30.0	29.0	47.2
Walker2d-expert	108.5 (10k)	0.9	56.0	**125.7**	107.1
Walker2d-medium	**74.6** (5k)	0.0	11.4	6.6	68.1
Halfcheetah-expert	84.4 (10k)	-1.7	105.2	**107.0**	88.1
Halfcheetah-medium	**39.0** (5k)	-3.1	36.6	36.1	**40.7**

Evaluation for KIO. Table 1 presents the final experimental results, where KIO achieves competitive scores in four out of six tasks. In these tasks, except for a slightly lower score in the Halfcheetah-expert task compared to the teacher agent, KIO's scores are either close to or exceed

[2]Regarding the definition of the score for one episode, we refer readers to the official documentation of Gymnasium [37] and the D4RL paper [17].

those of the teacher agent, indicating strong learning capabilities in complex control tasks. However, without the kernel method, the IO model demonstrates weak learning capabilities, achieving low scores in the Hopper task and failing to learn in the other two more challenging tasks. We argue that the weak performance of the IO model is due to the limitations of its hypothesis class, which lacks the richness needed to learn an effective policy for imitation learning tasks. This limitation arises from its reliance on predefined feature spaces, which may fail to capture the complexities of more sophisticated environments. All hyperparameters used in this experiment for KIO are listed in Appendix B.

Task	SCS		SSO	
	Obj Value	Score	Obj Value	Score
Hopper-expert	185.219	109.9	185.220	110.2
Hopper-medium	218.761	50.2	218.761	51.8
Walker2d-expert	140.121	108.5	140.121	109.2
Walker2d-medium	151.117	74.6	151.117	74.9
Halfcheetah-expert	165.041	84.4	165.041	83.8
Halfcheetah-medium	188.184	39.0	188.184	39.7

Table 2: Final Objective Function Value and Score (average return over 100 evaluations) for SCS [27] and SSO (20 iterations for all tasks) algorithms. The ultimate Objective Function Values of the two algorithms are nearly identical, yet across the majority of tasks, SSO achieves a slightly higher score compared to SCS.

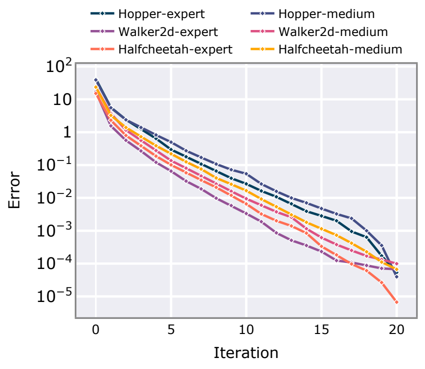

Figure 1: Convergence curves for SSO.

Evaluation for SSO. Table 2 presents the optimal objective function values and task scores obtained by the centralized algorithm, Splitting Conic Solver (SCS) [27], and the distributed algorithm, Sequential Selection Optimization (SSO), for solving Problem (9). In this experiment, SCS is employed to directly address the large-scale problem (9) with $|S| = N$. The corresponding solution is evaluated 100 times, and the average score is taken as the final result. Meanwhile, SSO addresses the large-scale problem by solving a series of subproblems. After each iteration, the current solution is evaluated 100 times, and the average score is recorded. After 20 iterations, there are 20 corresponding scores, and we select the highest score along with the objective function value from the last iteration and report it. The results show that SSO and SCS yield nearly identical optimal objective function values. However, except for the Halfcheetah-medium task, SSO achieved higher scores across all other tasks. Figure 1 displays the convergence performance of the SSO algorithm across six distinct tasks, with the horizontal axis representing the number of iterations and the vertical axis representing the error between the current objective function value and the optimal objective function value (calculated by SCS) in Problem (9). The SSO algorithm demonstrates a fast convergence rate. By the 10th iteration, the errors for all tasks are below 0.1, and by the 20th iteration, the errors have further diminished to approximately 1e-4 for all tasks.

In the previous evaluation of the SSO algorithm, we limited the maximum training data to 10k to ensure that we could directly solve Problem (9) without SSO. Thereby, we were able to compare the results with and without the SSO algorithm. However, to further verify the effectiveness of the SSO algorithm, we tested it on a new task (the medium-expert dataset) using 100k data points. At this scale, due to memory limitations, we were unable to solve Problem (9) directly without the SSO algorithm, thus the effectiveness of the SSO algorithm is inferred solely from its experimental performance. Table 3 presents the results of the KIO model optimized by the SSO algorithm. The results indicate that the KIO model achieves competitive results and scales effectively to larger data sizes. We list the hyperparameters used in this experiment in Appendix B.

5.2 Ablation Studies

We perform ablation studies to understand the contribution of each individual component: Heuristic Coordinates Selection (Section 4.1) and Warm-Up Trick (Section 4.2). Our results, presented in Figure 2, compare the performance of SSO with and without each component (all model hyperparameters remain unchanged as shown in Appendix B).

We use the Hopper task as the testing task, with the first 5000 data points from the D4RL Hopper-expert dataset as training data. The block coordinate for each iteration consists of

Table 3: Performance of KIO, two Behavior Cloning (BC) agents, and the Teacher agent on MuJoCo tasks from the D4RL benchmark on the normalized return metric. The numbers in parentheses represent the amount of data used by KIO, and the score for KIO in each task is the average score over 100 episodes.

Task	KIO	BC(TD3+BC)	BC(CQL)	Teacher
Hopper-medium-expert	79.6 (100k)	89.6	**111.9**	64.8
Walker2d-medium-expert	**100.1** (100k)	12.0	11.3	82.7
Halfcheetah-medium-expert	46.4 (100k)	**67.6**	35.8	64.4

2500 coordinates ($|S| = 2500$). When applying the Warm-Up Trick, we partition the data into two equal parts and solve two subproblems (9), each with $|S| = 2500$. Therefore, the computational time required for the Warm-Up Trick is approximately equal to the time needed for two iterations of the SSO algorithm. In Figure 2, we present the results of 20 iterations, with the vertical axis representing the difference between the current objective function value and the optimal value in Problem (9).

Both the Heuristic Coordinates Selection, abbreviated as Heuristic in Figure 2, and Warm-Up Trick significantly accelerate the algorithm. With the Warm-Up Trick, the initial objective function value is markedly reduced. Without the Warm-Up Trick, the Heuristic curve requires approximately 10 iterations to reach the initial values of the SSO curve, whereas the Warm-Up Trick requires only about the time of two iterations. The Heuristic Coordinates Selection results in rapid descent of the error curves. The WarmUp curve, however, becomes nearly flat after a few iterations, until the 17th iteration when the Heuristic Coordinates Selection method is activated, causing a rapid decrease in the curve.

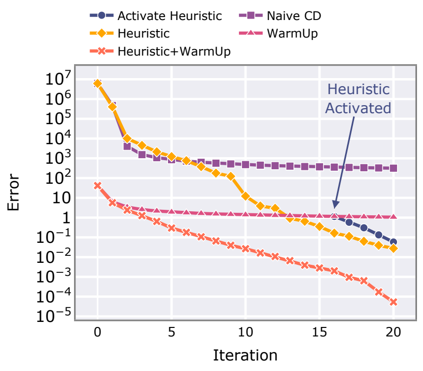

Figure 2: Convergence curves on the MuJoCo Hopper task with the first 5k data points from the D4RL Hopper-expert dataset. The vertical axis represents the difference between the current objective function value and the optimal value. Sequential Selection Optimization (orange) exhibits the fastest convergence rate.

6 Conclusion and Limitations

We introduced Kernel Inverse Optimization (KIO), an inverse optimization model leveraging kernel methods, along with its simplified variant and the theoretical derivations. Subsequently, we proposed the Sequential Selection Optimization (SSO) algorithm for training the KIO model, which addresses memory issues by decomposing the original problem into a series of subproblems. Our empirical results demonstrate that KIO exhibits strong learning capabilities in complex control tasks, while the SSO algorithm achieves rapid convergence to the optimal solution within a limited number of iterations.

One of the limitations of this model is the computational cost of adding a new data point. In that case, all training data are required to compute the coefficients for the FOP problem (see FOP (8)) for that point. Thus, as the amount of training data grows, so does the computational cost. Another limitation is the absence of theoretical analysis on the convergence rate of the SSO algorithm, which we leave for future research. Furthermore, in our numerical experiments in Section 5, the proposed KIO model, even with the SSO algorithm, required substantial memory resources—up to 256 GB when using 100k data points. However, we believe that further optimization in implementation could reduce these memory requirements. Finally, in our numerical experiments in Section 5, we observed that initialization strategies critically impact the performance of the SSO algorithm. Thus, exploring alternative initialization strategies beyond the one proposed in Section 4.2 presents a promising direction for future work.

SUPPLEMENTARY

A Technical Proofs

A.1 Proof of Theorem 1

First, let P, $\tilde{\lambda}_i$ and $\begin{bmatrix}\Lambda_i & \Gamma_i \\ * & \alpha_i\end{bmatrix}$ be the Lagrange multiplier associated with the constraints $\theta_{uu} \succeq I_m$, $\lambda_i \in \mathbb{R}^d_+$ and $\begin{bmatrix}\theta_{uu} & \hat{M}_i^\top \lambda_i + 2\theta_{su}^\top \phi(\hat{s}_i) \\ * & \gamma_i\end{bmatrix} \succeq 0$ respectively, where $P, \Lambda_i \in \mathbb{R}^{n\times n}$, $\Gamma_i \in \mathbb{R}^n$, $\tilde{\lambda}_i \in \mathbb{R}^d$ and $\gamma_i \in \mathbb{R}$. Then define the Lagrangian function

$$
\begin{aligned}
L(\theta_{uu}, \theta_{su}, \lambda_i, \gamma_i, P, \tilde{\lambda}_i, \Lambda_i, \Gamma_i, \alpha_i) =& k\|\theta_{uu}\|_F^2 + k\|\theta_{su}\|_F^2 - \mathrm{Tr}(P(\theta_{uu} - I_n)) - \sum_{i=1}^N \tilde{\lambda}_i^\top \lambda_i \\
&+ \frac{1}{N}\sum_{i=1}^N \left(\hat{u}_i^\top \theta_{uu}\hat{u}_i + 2\phi(\hat{s}_i)^\top \theta_{su}\hat{u}_i + \frac{1}{4}\gamma_i + \hat{W}_i^\top \lambda_i\right) \\
&- \sum_{i=1}^N \mathrm{Tr}\left(\begin{bmatrix}\Lambda_i & \Gamma_i \\ * & \alpha_i\end{bmatrix}\begin{bmatrix}\theta_{uu} & \hat{M}_i^\top \lambda_i + 2\theta_{su}^\top \phi(\hat{s}_i) \\ * & \gamma_i\end{bmatrix}\right),
\end{aligned} \tag{13}
$$

and the Lagrange dual problem

$$
\begin{array}{ll}
\max\limits_{P,\tilde{\lambda}_i,\Lambda_i,\Gamma_i,\alpha_i} & \inf\limits_{\theta_{uu},\theta_{su},\lambda_i,\gamma_i} L(\theta_{uu}, \theta_{su}, \lambda_i, \gamma_i, P, \tilde{\lambda}_i, \Lambda_i, \Gamma_i, \alpha_i) \\
\text{s.t.} & P \succeq 0,\ \tilde{\lambda}_i \in R^d_+, \quad \forall i \le N \\
& \begin{bmatrix}\Lambda_i & \Gamma_i \\ * & \alpha_i\end{bmatrix} \succeq 0, \quad \forall i \le N.
\end{array} \tag{14}
$$

When the Lagrangian function (13) is at the point of the infimum with respect to $\theta_{uu}, \theta_{su}, \lambda_i, \gamma_i$, we have

$$
\begin{aligned}
\frac{\partial L}{\partial \theta_{uu}} &= 2k\theta_{uu} + \frac{1}{N}\sum_{i=1}^N \hat{u}_i\hat{u}_i^\top - P + \sum_{i=1}^N -\Lambda_i = 0 \Rightarrow \theta_{uu} = \frac{1}{2k}\left(P - \left(\sum_{i=1}^N \frac{\hat{u}_i\hat{u}_i^\top}{N} - \Lambda_i\right)\right) \\
\frac{\partial L}{\partial \theta_{su}} &= 2k\theta_{su} + 2\sum_{i=1}^N \phi(\hat{s}_i)\left(\frac{\hat{u}_i^\top}{N} - 2\Gamma_i^\top\right) = 0 \Rightarrow \theta_{su} = \sum_{i=1}^N \phi(\hat{s}_i)\frac{1}{k}\left(2\Gamma_i^\top - \frac{\hat{u}_i^\top}{N}\right) \\
\frac{\partial L}{\partial \lambda_i} &= \frac{\hat{W}_i}{N} - \tilde{\lambda}_i - 2\hat{M}_i\Gamma_i = 0 \Rightarrow \tilde{\lambda}_i = \frac{\hat{W}_i}{N} - 2\hat{M}_i\Gamma_i \\
\frac{\partial L}{\partial \gamma_i} &= \frac{1}{4N} - \alpha_i = 0 \Rightarrow \alpha_i = \frac{1}{4N}
\end{aligned}
$$

Finally, substituting the expressions for θ_{uu}, θ_{su}, $\tilde{\lambda}_i$, and α_i into the Lagrange dual problem (14) and simplifying completes the proof.

A.2 Proof of Proposition 2

Note that Problem (9) is the dual problem of Problem (4) with $\theta_{uu} = I_n$. Here, we enumerate the five KKT conditions that will be employed

$$\theta_{su} = \frac{1}{k}\sum_{i=1}^N \phi(\hat{s}_i)\left(2\Gamma_i^\top - \frac{\hat{u}_i^\top}{N}\right), \qquad \text{(stationarity)} \tag{15a}$$

$$\tilde{\lambda}_i = \frac{\hat{W}_i}{N} - 2\hat{M}_i\Gamma_i,\ \forall i \le N, \qquad \text{(stationarity)} \tag{15b}$$

$$\tilde{\lambda}_i^\top \lambda_i = 0,\ \forall i \le N, \qquad \text{(complementary slackness)} \tag{15c}$$

$$\mathrm{Tr}\left(\begin{bmatrix}\Lambda_i & \Gamma_i \\ * & \frac{1}{4N}\end{bmatrix}\begin{bmatrix}I_n & \hat{M}_i^\top \lambda_i + 2\theta_{su}^\top \phi(\hat{s}_i) \\ * & \gamma_i\end{bmatrix}\right) = 0,\ \forall i \le N, \qquad \text{(complementary slackness)} \tag{15d}$$

$$\lambda_i \in R^d_+,\ \forall i \le N. \qquad \text{(primal feasibility)} \tag{15e}$$

First, we choose coordinate i that satisfy Condition (10). Then based on KKT condition (15b), we have

$$\tilde{\lambda}_i > 0. \tag{16}$$

Next, based on conditions in (15c) and (15e), one can obtain

$$\lambda_i = 0. \tag{17}$$

Substituting the result (17) into Condition (15d) yields

$$\mathrm{Tr}\left(\begin{bmatrix}\Lambda_i & \Gamma_i \\ * & \frac{1}{4N}\end{bmatrix}\begin{bmatrix}I_n & 2\theta_{su}^\top \phi(\hat{s}_i) \\ * & \gamma_i\end{bmatrix}\right) = 0. \tag{18}$$

γ_i is the decision variable of Problem (4) with $\theta_{uu} = I_n$. Next, let's solve for its expression. By utilizing the Schur complement, we can prove the following two constraints are equivalent

$$\begin{bmatrix}I_n & \hat{M}_i^\top \lambda_i + 2\theta_{su}^\top \phi(\hat{s}_i) \\ * & \gamma_i\end{bmatrix} \succeq 0 \Leftrightarrow \gamma_i \geq \left\|\hat{M}_i^\top \lambda_i + 2\theta_{su}^\top \phi(\hat{s}_i)\right\|_2^2.$$

Therefore, Problem (4) with $\theta_{uu} = I_n$ can be equivalently expressed as

$$\begin{aligned}\min_{\theta_{su},\gamma_i,\lambda_i} \quad & k\|\theta_{su}\|_F^2 + \frac{1}{N}\sum_{i=1}^N \left(2\hat{u}_i^T \theta_{su}^T \phi(\hat{s}_i) + \frac{1}{4}\gamma_i + \hat{W}_i^\top \lambda_i\right) \\ \text{s.t.} \quad & \lambda_i \in \mathbb{R}_+^d, \ \gamma_i \in \mathbb{R}, \quad \forall i \leq N \\ & \gamma_i \geq \left\|\hat{M}_i^\top \lambda_i + 2\theta_{su}^\top \phi(\hat{s}_i)\right\|_2^2, \quad \forall i \leq N,\end{aligned} \tag{19}$$

Here, the variable γ_i is highlighted. It can be easily proven that when γ_i attain its optimal values, the equality in the last constraint should hold: $\gamma_i = \left\|\hat{M}_i^\top \lambda_i + 2\theta_{su}^\top \phi(\hat{s}_i)\right\|_2^2$. Note that $\lambda_i = 0$ (17), then the expression of γ_i is

$$\gamma_i = \left\|2\theta_{su}^\top \phi(\hat{s}_i)\right\|_2^2. \tag{20}$$

Substituting (20) into (18), one can obtain Condition (11).

B Hyperparameters of KIO

Table 4 provides the hyperparameters used in the experiments in Section 5.

Table 4: KIO Environment Specific Parameters. N is the size of the dataset, `p` is the number of coordinates that are updated in one iteration of SSO, k is the regularization coefficient, and `scalar` is the multiplier used in the implementation to avoid numerical instabilities.

Environment	Dataset	k	`scalar`	`p`
Hopper-expert	0:5k	1e-6	$200N$	$0.5N$
Hopper-medium	0:5k	1e-6	$200N$	$0.5N$
Hopper-medium-expert	First 50k + Last 50k	1e-6	$200N$	$0.1N$
walker2d-expert	0:5k+10k:15k	1e-5	$50N$	$0.5N$
walker2d-medium	15k:20k	1e-5	$50N$	$0.5N$
walker2d-medium-expert	First 50k + Last 50k	1e-5	$50N$	$0.1N$
Halfcheetah-expert	325k:330k+345k:350k	5e-6	$50N$	$0.5N$
Halfcheetah-medium	10k:15k	5e-6	$50N$	$0.5N$
Halfcheetah-medium-expert	First 50k + Last 50k	1e-6	$50N$	$0.1N$

C Ablation Study on Kernel Function

Table 5: Performance of KIO on MuJoCo tasks from the D4RL benchmark on the normalized return metric. The scores in each task represent the average score over 100 episodes within the range of one standard deviation.

Task	**RBF**	**Laplace**	**Linear**
Hopper-medium	51 ± 6.4	41 ± 5.1	24 ± 0.05
Hopper-expert	109.9 ± 0.4	71 ± 26.7	28 ± 0.5
Walker2d-medium	72 ± 14	43 ± 28.2	-0.19 ± 0.006
Walker2d-expert	109.1 ± 0.3	103.1 ± 22.6	-0.02 ± 0.1
Halfcheetah-medium	32.4 ± 12.2	52 ± 10.5	-0.8 ± 0.6
Halfcheetah-expert	78.8 ± 24.9	59.1 ± 35.5	2.0 ± 3.1

In addition to the studies presented in Section 5.2, we conducted experiments to evaluate our proposed model using different kernels commonly employed in machine learning: Gaussian (RBF), Laplacian, and linear kernels. The results for the RBF kernel in Table 5 differ slightly from those in Table 3 due to using a different dataset partition for normalizing the states.

Acknowledgments

This work is partially supported by the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation programme (TRUST-949796).

References

[1] Ravindra K Ahuja and James B Orlin. Inverse optimization. *Operations research*, 49(5):771–783, 2001.

[2] Syed Adnan Akhtar, Arman Sharifi Kolarijani, and Peyman Mohajerin Esfahani. Learning for control: An inverse optimization approach. *IEEE Control Systems Letters*, 6:187–192, 2021.

[3] Mosek ApS. Mosek optimization toolbox for matlab. *User's Guide and Reference Manual, Version*, 4:1, 2019.

[4] Anil Aswani, Zuo-Jun Shen, and Auyon Siddiq. Inverse optimization with noisy data. *Operations Research*, 66(3):870–892, 2018.

[5] Turgay Ayer. Inverse optimization for assessing emerging technologies in breast cancer screening. *Annals of operations research*, 230:57–85, 2015.

[6] Gaston Baudat and Fatiha Anouar. Generalized discriminant analysis using a kernel approach. *Neural computation*, 12(10):2385–2404, 2000.

[7] Dimitri Bertsekas. *Convex optimization algorithms*. Athena Scientific, 2015.

[8] Dimitris Bertsimas, Vishal Gupta, and Ioannis Ch Paschalidis. Data-driven estimation in equilibrium using inverse optimization. *Mathematical Programming*, 153:595–633, 2015.

[9] John R Birge, Ali Hortaçsu, and J Michael Pavlin. Inverse optimization for the recovery of market structure from market outcomes: An application to the miso electricity market. *Operations Research*, 65(4):837–855, 2017.

[10] Timothy CY Chan, Rafid Mahmood, and Ian Yihang Zhu. Inverse optimization: Theory and applications. *Operations Research*, 2023.

[11] Lu Chen, Yuyi Chen, and André Langevin. An inverse optimization approach for a capacitated vehicle routing problem. *European Journal of Operational Research*, 295(3):1087–1098, 2021.

[12] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20:273–297, 1995.

[13] Steven Diamond and Stephen Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.

[14] András Faragó, Áron Szentesi, and Balázs Szviatovszki. Inverse optimization in high-speed networks. *Discrete Applied Mathematics*, 129(1):83–98, 2003.

[15] Ricardo Fernández-Blanco, Juan Miguel Morales, and Salvador Pineda. Forecasting the price-response of a pool of buildings via homothetic inverse optimization. *Applied Energy*, 290:116791, 2021.

[16] Ricardo Fernández-Blanco, Juan Miguel Morales, Salvador Pineda, and Álvaro Porras. Inverse optimization with kernel regression: Application to the power forecasting and bidding of a fleet of electric vehicles. *Computers & Operations Research*, 134:105405, 2021.

[17] Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning, 2020.

[18] Scott Fujimoto and Shixiang Shane Gu. A minimalist approach to offline reinforcement learning. *Advances in neural information processing systems*, 34:20132–20145, 2021.

[19] Garud Iyengar and Wanmo Kang. Inverse conic programming with applications. *Operations Research Letters*, 33(3):319–330, 2005.

[20] Arezou Keshavarz, Yang Wang, and Stephen Boyd. Imputing a convex objective function. In *2011 IEEE international symposium on intelligent control*, pages 613–619. IEEE, 2011.

[21] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 33:1179–1191, 2020.

[22] Jonathan Yu-Meng Li. Inverse optimization of convex risk functions. *Management Science*, 67(11):7113–7141, 2021.

[23] Charles A Micchelli and Massimiliano Pontil. On learning vector-valued functions. *Neural computation*, 17(1):177–204, 2005.

[24] Peyman Mohajerin Esfahani and Daniel Kuhn. Data-driven distributionally robust optimization using the wasserstein metric: performance guarantees and tractable reformulations. *Mathematical Programming*, 171(1-2):115–166, 2018.

[25] Kevin P. Murphy. *Probabilistic machine learning: an introduction*. MIT press, 2022.

[26] Yu Nesterov. Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM Journal on Optimization*, 22(2):341–362, 2012.

[27] Brendan O'Donoghue, Eric Chu, Neal Parikh, and Stephen Boyd. Conic optimization via operator splitting and homogeneous self-dual embedding. *Journal of Optimization Theory and Applications*, 169(3):1042–1068, June 2016.

[28] Michael Patriksson. *The traffic assignment problem: models and methods*. Courier Dover Publications, 2015.

[29] John Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. Technical report, Microsoft, 1998.

[30] Mikael Rönnqvist, Gunnar Svenson, Patrik Flisberg, and Lars-Erik Jönsson. Calibrated route finder: Improving the safety, environmental consciousness, and cost effectiveness of truck routing in sweden. *Interfaces*, 47(5):372–395, 2017.

[31] Javier Saez-Gallego, Juan M Morales, Marco Zugno, and Henrik Madsen. A data-driven bidding model for a cluster of price-responsive consumers of electricity. *IEEE Transactions on Power Systems*, 31(6):5001–5011, 2016.

[32] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Kernel principal component analysis. In *International conference on artificial neural networks*, pages 583–588. Springer, 1997.

[33] Bernhard Scholkopf and Alexander J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, Cambridge, MA, USA, 2001.

[34] Soroosh Shafieezadeh-Abadeh, Daniel Kuhn, and Peyman Mohajerin Esfahani. Regularization via mass transportation. *Journal of Machine Learning Research*, 20(103):1–68, 2019.

[35] Zahed Shahmoradi and Taewoo Lee. Quantile inverse optimization: Improving stability in inverse linear programming. *Operations research*, 70(4):2538–2562, 2022.

[36] Hao-Jun Michael Shi, Shenyinying Tu, Yangyang Xu, and Wotao Yin. A primer on coordinate descent algorithms. *arXiv preprint arXiv:1610.00040*, 2016.

[37] Mark Towers, Jordan K. Terry, Ariel Kwiatkowski, John U. Balis, Gianluca de Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Arjun KG, Markus Krimmel, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Andrew Tan Jin Shen, and Omar G. Younis. Gymnasium, March 2023.

[38] Shi Yu, Haoran Wang, and Chaosheng Dong. Learning risk preferences from investment portfolios using inverse optimization. *Research in International Business and Finance*, 64:101879, 2023.

[39] Pedro Zattoni Scroccaro, Bilge Atasoy, and Peyman Mohajerin Esfahani. Learning in inverse optimization: Incenter cost, augmented suboptimality loss, and algorithms. *Operations Research*, 2024.

[40] Pedro Zattoni Scroccaro, Piet van Beek, Peyman Mohajerin Esfahani, and Bilge Atasoy. Inverse optimization for routing problems. *Transportation Science*, 2024.