Incorporating Linear Dependencies into Graph Gaussian Processes

by

Yueheng Zhang

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2023

Statement of Contributions

Chapter 1, Chapter 2.2, Chapter 3, and Chapter 4 are based on a conference submission with coauthors Alexander Terenin, Haolin Yu, and Prof. Pascal Poupart, where I was the lead author. Alexander Terenin simplified the presentation of our construction in Chapter 3.1.1, provided expertise on existing graph kernels, and gave valuable suggestions for texts and figures. Haolin Yu provided helpful suggestions for technical details in the reinforcement learning experiment. This entire work was conducted in collaboration with Prof. Pascal Poupart, who provided general guidance.

Abstract

Graph Gaussian processes are an important technique for learning unknown functions on graphs while quantifying uncertainty. These processes encode prior information by using kernels that reflect the structure of the graph, allowing function values at nearby nodes to be correlated. However, there are limited choices for kernels on graphs, and most existing graph kernels can be shown to rely on the graph Laplacian and behave in a manner that resembles Euclidean radial basis functions.

In many applications, additional prior information which goes beyond the graph structure encoded in Laplacian is available: in this work, we study the case where the dependencies between nodes in the target function are known as linear, possibly up to some noise. We propose a type of kernel for graph Gaussian processes that incorporate linear dependencies between nodes, based on an inter-domain-type construction. We show that this construction results in kernels that can encode directed information, and are robust under misspecified linear dependencies. We also show that the graph Matérn kernel, one of the commonly used Laplacian-based kernels, can be obtained as a special case of this construction.

We illustrate the properties of these kernels on a set of synthetic examples. We then evaluate these kernels in a real-world traffic speed prediction task, and show that they easily out-perform the baseline kernels. We also use these kernels to learn offline reinforcement learning policies in maze environments. We show that they are significantly more stable and data-efficient than strong baselines, and they can incorporate prior information to generalize to unseen tasks.

Acknowledgements

I would like to thank Prof. Lap Chi Lau for supervising me in the first year of my program, and for supporting me in my exploration of different areas which has been an invaluable experience for me. I am grateful to Prof. Pascal Poupart for his guidance throughout this project, and for believing in me and taking the time with me so that this project could take shape starting from a course project. I thank my coauthors Alexander Terenin and Haolin Yu for their suggestions and insights on both our project at hand and research methods in general. I thank my thesis readers Prof. Kimon Fountoulakis and Prof. Yaoliang Yu for reading my thesis and providing helpful feedback. I would also like to thank Prof. Wenhu Chen for his time and advice when I was exploring research topics.

I would like to thank my friends, labmates, and fellow students in Waterloo, whose company I value and enjoy. I thank my parents for always supporting me no matter what I choose to do. Thank you to my partner Marian, for always encouraging and supporting me even when I doubt myself, and for being my belayer so that I dare to reach and jump.

Table of Contents

List of Figures

Chapter 1

Introduction

Graphs are a ubiquitous data structure that represents relations between discrete entities, such as how people interact in social networks [27], how places are connected in road networks [21], how molecules interact in biology networks [17], etc.

Functions on graphs give additional information for the relations they represent [1]. For example, in a social network, a binary function on the people can designate which people are in a certain club; in a road network, a function mapping edges to the reals can store distances between places; in a time series of gene regulation graphs, a function mapping graphs to integers can tell us which time point a gene-interaction graph corresponds to. Machine learning on graphs aims to approximate such functions to predict function values at given inputs.

As in the above examples, the input space to the functions can be a set of nodes, a set of edges, or entire graphs, and the function output can come from discrete classes as in classification tasks, or from a continuous range as in regression problems. In this work, we tackle the specific problem of regression on functions defined on the node set of a given graph.

1.1 Problem statement

More precisely, our setup is as follows. Given a graph $G = (V, E)$ with node set V, edge set $E \in V \times V$ and edge weight function $w : E \to \mathbb{R}$, we assume that there exists an unknown real-valued target function $f : V \to \mathbb{R}$ on the node set. Given data $\mathcal{D} = \{(v_i, f(v_i))\}_i$

consisting of input-output pairs for f, where $v_i \in V$, we would like to predict $f(v)$ for any $v \in V$.

One of the major challenges of machine learning on graphs is the irregular input spaces, which prevents direct applications of machine learning methods successful in Euclidean spaces [4]. Graph neural networks (GNNs), a category of deep neural networks specifically designed for graph-structured data, have been applied to a wide range of machine learning problems on graphs [46]. However, certain properties of GNNs might not be desirable in all applications. For example, explaining the decision process of GNNs is challenging, and methods for this purpose are usually either heuristic themselves, or complicated and computationally expensive [56]. Options to quantify the uncertainty during the decision process of GNNs are also limited [54]. Our work therefore explores using Gaussian processes for learning on graphs as an alternative to GNNs.

Gaussian processes are a versatile technique for approximating and quantifying uncertainty about unknown functions [38], making them especially suitable for safety-critical tasks. A Gaussian process can be seen as a random function from the input space to the reals, where uncertainty for values at observed inputs are low and those at unobserved inputs are high. The key ingredient in designing a Gaussian process regression model is the covariance function, also known as the kernel, which specifies how correlated the values at two input points are.

While there is a large literature on how to design kernels on $\mathbb{R}^d$ [38], techniques for graphs are comparatively less-developed. Radial basis functions are a popular class of kernels in Euclidean space, where the covariance between two points is only dependent on the distance between them. However, simply replacing the Euclidean distance by graph distance do not always result in well-defined kernels [3, 13], again because of the different geometry. Most exisiting graph kernels can be shown to rely on the graph Laplacian [22, 39], and behave in a manner that resembles Euclidean radial basis functions.

In many applications on graphs, more information is available than the structural information encoded in the Laplacian matrix. For example, we might not only know that the function value at one node is closely related to the values of its neighbors, but also *how* its value relates to those of its neighbors [6][41][7][5]. In such cases, it is natural to seek to incorporate this additional prior information into the construction of the kernel, in order to achieve better prediction accuracy and data-efficiency.

1.2 Functions with linear dependencies on graphs

We focus on the special case where the ground-truth unknown function $f : V \to \mathbb{R}$ has the property that the value at one node is a linear combination of the value of its neighbors, potentially up to some noise. To ease exposition, we begin with the noiseless case. Specifically, we say that the unknown function f *satisfies linear constraints* if for all $i \in V$

$$f(i) = \sum_{j \in V, (i,j) \in E} m_{ij} f(j) \tag{1.1}$$

for some set of problem-dependent coefficients m_{ij} that define the constraints. If we reinterpret the function $f : V \to \mathbb{R}$ as a vector $\boldsymbol{f} \in \mathbb{R}^{|V|}$, this means that $\boldsymbol{f} = \mathbf{M}\boldsymbol{f}$, where $\mathbf{M}$ is the matrix defined by the coefficients: hence, $\boldsymbol{f}$ must lie in the span of the unit-eigenvalue eigenvectors of $\mathbf{M}$. Exact linear constraints are therefore a fairly rigid notion: we will generally be interested in cases where (1.1) holds approximately, possibly due to incomplete observations of f, modeling simplifications, or the presence of noise: we develop this further in Section 3.1. We note that this notion allows for *directed* constraints, even if the underlying graph is undirected—we will make use of this flexibility in later sections. Figure 1.1 visualizes two examples of linear dependencies on graphs.

To better motivate the setup of (1.1), note that if $\mathbf{M}$ is a stochastic matrix, then its entries correspond exactly to the transition probabilities of a discrete-time Markov chain over a finite state space. In this case, (1.1) are the equations which define the chain's stationary distribution. Such Markov chains are used to model many real-world dynamics where the values of $\mathbf{M}$ are based on modeling assumptions. In these cases, one might be interested in inferring $\boldsymbol{f}$ based on a combination of these assumptions and its observed values based on data.

Our problem specification encompasses a wide range of applications. Graphs with linear dependencies arise in linear Gaussian structural equation models—a common type of model for inferring causal effects [6]. Our formulation also applies to Bellman equations from control and reinforcement learning, which we will illustrate further in Section 4.2. In addition, certain mesh editing operations based on discretized Laplacian commonly used in computer graphics have the same linear form when viewed appropriately [41].

For problems where we know the value at one node is a function of the value of its neighbors, but the exact relations are unknown or too complex, incorporating a simplified set of linear constraints, possibly based on Taylor approximations, together with noise, can be a simple and effective choice. Examples of this include modeling of complex dynamics on graphs such as those arising in energy optimization, where locally linear assumptions

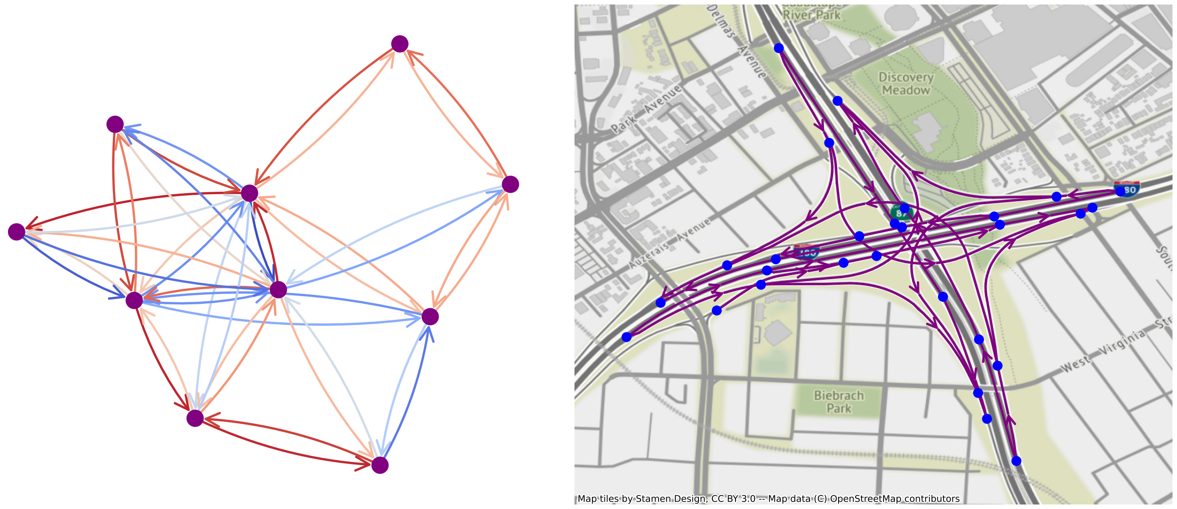

Figure 1.1: Left: illustration of linear constraints on a randomly-drawn synthetic graph. A red edge corresponds to a constraint with positive coefficient, and a blue edge corresponds to a constraint with negative coefficient. A more-solid line corresponds to a greater coefficient in absolute value. Right: illustration of linear constraints according to directions of traffic flow on a road network, where nodes are traffic sensors – more details can be found in Section 4.1. In both figures, the node at the head of the directed edge depends linearly on the node at the tail.

are used to approximate building thermodynamics [7]. Another example is traffic speed estimation on road networks: although a comprehensive modeling framework might involve many non-linear terms, we illustrate in Section 4.1 that simple linear constraints derived from first principles can improve performance when incorporated into a speed prediction task.

In all of the examples mentioned above, we have a target function $f : V \to \mathbb{R}$ on graphs that we assume to have linear dependencies between a node and its neighbors, possibly up to some noise. Our task is to predict unknown values of f from the graph, using a model of the structure of the linear dependencies, and noisy observations of the value of f for certain nodes.

1.3 Contribution

In this work, we propose a type of graph kernels which incorporates linear dependencies on a graph. This is done through a simple and easy-to-interpret interdomain-type construction. This kernel can be used both in cases where dependencies are known exactly, and in cases where they are known approximately up to some noise. Our theoretical and empirical contributions are as follows.

- We examine the prior information these kernels include, analyze their robustness under misspecification, and show their connection to Laplacian based kernels. We illustrate their properties on a set of synthetic examples.
- We evaluate these kernels in a real-world traffic speed prediction task, and show that they easily out-perform the baseline kernels.
- We use these kernels to learn offline reinforcement learning (RL) policies in maze environments. We show that they are significantly more stable and data-efficient than strong baselines, and they can incorporate prior information to generalize to unseen tasks.

Chapter 2

Background

In this chapter, we introduce necessary technical background, as well as related research areas and prior works. Specifically, we derive the basics of Gaussian processes, and then give an overview for related works on graph kernels and constrained Gaussian processes. Since reinforcement learning is a major application of this work, we also introduce the problem setup for reinforcement learning. Section 2.1 references [38], and Section 2.4 references [44].

2.1 Kernels and Gaussian processes

We begin by reviewing properties of multivariate Gaussian distribution. We then introduce Bayesian linear regression and Gaussian proceses, and show their connections to each other.

Throughout this section, we use lower case letters for scalars, bold lower case letters for vectors, and bold upper case letters for matrices.

2.1.1 Properties of multivariate Gaussians

A multivariate Gaussian random vector $\boldsymbol{x} \in \mathbb{R}^n$ with mean $\boldsymbol{\mu} \in \mathbb{R}^n$ and $n \times n$ covariance matrix $\boldsymbol{\Sigma}$, written as

$$\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$$

has joint probability density function

$$p(\boldsymbol{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}) = (2\pi)^{-n/2}|\boldsymbol{\Sigma}|^{-1/2} \exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right). \tag{2.1}$$

Let $\boldsymbol{y} := \mathbf{A}\boldsymbol{x} + \boldsymbol{b}$ be an affine transformation of $\boldsymbol{x}$ with $A \in \mathbb{R}^{m\times n}$ and $\boldsymbol{b} \in \mathbb{R}^m$. Then, $\boldsymbol{y}$ is an m-dimensional multivariate Gaussian as follows:

$$\boldsymbol{y} \mid \boldsymbol{x}, \mathbf{A}, \boldsymbol{b} \sim \mathcal{N}(\mathbf{A}\boldsymbol{\mu} + \boldsymbol{b}, \quad \mathbf{A}\boldsymbol{\Sigma}\mathbf{A}^T). \tag{2.2}$$

Let $\boldsymbol{x}_1$, $\boldsymbol{x}_2$ be jointly Gaussian:

$$\begin{bmatrix}\boldsymbol{x}_1\\ \boldsymbol{x}_2\end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix}\boldsymbol{\mu}_1\\ \boldsymbol{\mu}_2\end{bmatrix}, \begin{bmatrix}\boldsymbol{\Sigma}_1 & \boldsymbol{\Sigma}_{12}\\ \boldsymbol{\Sigma}_{12}^T & \boldsymbol{\Sigma}_2\end{bmatrix}\right).$$

Then, the marginal distribution of $\boldsymbol{x}_1$ is

$$\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_2). \tag{2.3}$$

The conditional distribution of $\boldsymbol{x}_1$ given $\boldsymbol{x}_2$ is

$$\boldsymbol{x}_1 \mid \boldsymbol{x}_2 \sim \mathcal{N}(\boldsymbol{\mu}_1 + \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_2^{-1}(\boldsymbol{x}_2 - \boldsymbol{\mu}_2), \quad \boldsymbol{\Sigma}_1 - \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_2^{-1}\boldsymbol{\Sigma}_{12}^T). \tag{2.4}$$

Rigorous definitions of multivariate Gaussian and proofs of the above properties can be found in [15].

2.1.2 Bayesian linear regression

Let $f : \mathbb{R}^n \to \mathbb{R}$ be a target function, and $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_i^N$ be input-output pairs observed from f where $\boldsymbol{x}_i \in \mathbb{R}^n$ and y_i are scalars. We want to learn a regression function f_* from $\mathcal{D}$ such that f_* approximates f.

Let $\boldsymbol{\phi} : \mathbb{R}^n \to \mathbb{R}^d$ be a feature map that takes inputs $\boldsymbol{x} \in \mathbb{R}^n$ to a feature space in $\mathbb{R}^d$. For example, when $\boldsymbol{\phi}$ is the identity, input space and feature space coincide. Another example is when x is a scalar, and $\boldsymbol{\phi}(x) = (1, x, x^2, \ldots, x^k)^T$. This enables polynomial regression with respect to the input x by linear regression in the feature space.

Linear regression models assume that the target function f is linear with respect to the features $\boldsymbol{\phi}(\boldsymbol{x})$, and we observe f up to noises. More precisely, for a given datapoint $(\boldsymbol{x}, y)$, we assume that

$$f(\boldsymbol{x}) = \boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{w}, \quad y = f(\boldsymbol{x}) + \epsilon(\boldsymbol{x}). \tag{2.5}$$

Here $\epsilon(\boldsymbol{x})$ is the noise term, which is often modeled as i.i.d. Gaussian random variables with zero mean and variance σ_ϵ. The model fitting process consists of learning the paramters $\boldsymbol{w}$ and arriving at the regression function $f_*(\cdot) := \boldsymbol{\phi}(\cdot)^T \boldsymbol{w}$. We then use f_* to predict the value of f on new inputs $\boldsymbol{x}_{*1}, \ldots, \boldsymbol{x}_{*M}$.

For compact notation, we write the N datapoints in $\mathcal{D}$ together in vector notation. We write $\boldsymbol{y}$ for $(y_1, \ldots, y_N)$. We write $\mathbf{X}$ for the $n \times N$ matrix where columns are $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$, write $\mathbf{\Phi}$ for the $d \times N$ matrix where columns are $\boldsymbol{\phi}(\boldsymbol{x}_1), \ldots, \boldsymbol{\phi}(\boldsymbol{x}_n)$, and likewise for $\mathbf{X}_*$ and $\mathbf{\Phi}_*$.

Then, applying our model in Equation (2.5) to all datapoints in $\mathcal{D}$, we have that

$$\boldsymbol{y} = \mathbf{\Phi}^T \boldsymbol{w} + \boldsymbol{\epsilon}, \quad \text{where } \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \sigma_\epsilon^2 \mathbf{I}). \tag{2.6}$$

To learn the parameters $\boldsymbol{w}$ from $\mathcal{D}$ in a Bayesian way, we view $\boldsymbol{w}$ as a multivariate Gaussian random vector with zero mean and covariance function $\mathbf{\Sigma}$:

$$\boldsymbol{w} \sim \mathcal{N}(\mathbf{0}, \mathbf{\Sigma}).$$

Bayes' rule states that

$$\text{posterior} = \frac{\text{likelihood} \cdot \text{prior}}{\text{marginal likelihood}}.$$

In our case, the posterior distribution for the parameters can be expressed as

$$p(\boldsymbol{w} \mid \boldsymbol{y}, \mathbf{\Phi}) = \frac{p(\boldsymbol{y} \mid \boldsymbol{w}, \mathbf{\Phi}) p(\boldsymbol{w})}{p(\boldsymbol{y} \mid \mathbf{\Phi})} = \frac{p(\boldsymbol{y} \mid \boldsymbol{w}, \mathbf{\Phi}) p(\boldsymbol{w})}{\int p(\boldsymbol{y} \mid \boldsymbol{w}, \mathbf{\Phi}) p(\boldsymbol{w}) d(\boldsymbol{w})}. \tag{2.7}$$

To obtain the left-hand side, we need to find an expression for $p(\boldsymbol{y} \mid \boldsymbol{w}, \mathbf{\Phi})$. In Equation (2.6), when $\mathbf{\Phi}$ and $\boldsymbol{w}$ are given, $\boldsymbol{y}$ is a multivariate Gaussian $\boldsymbol{\epsilon}$ shifted by a constant vector and thus a multivariate Gaussian itself by (2.2), with distribution

$$p(\boldsymbol{y} \mid \boldsymbol{w}, \mathbf{\Phi}) = \mathcal{N}(\mathbf{\Phi}^T \boldsymbol{w}, \sigma_\epsilon^2 \mathbf{I}).$$

Then, expanding the density function (2.1) and completing the squares, the variables in the numerator in Equation (2.7) becomes

$$\begin{aligned} p(\boldsymbol{y} \mid \boldsymbol{w}, \boldsymbol{\Phi}) p(\boldsymbol{w}) &\propto \exp\left(-\frac{1}{2\sigma_\epsilon^2}(\boldsymbol{y} - \boldsymbol{\Phi}^T \boldsymbol{w})^T (\boldsymbol{y} - \boldsymbol{\Phi}^T \boldsymbol{w})\right) \exp\left(-\frac{1}{2}\boldsymbol{w}^T \boldsymbol{\Sigma}^{-1} \boldsymbol{w}\right) \\ &\propto \exp\left(-\frac{1}{2}(\boldsymbol{w} - \mathbf{A}\boldsymbol{\Phi}\boldsymbol{y}/\sigma_\epsilon^2)^T \mathbf{A}^{-1} (\boldsymbol{w} - \mathbf{A}\boldsymbol{\Phi}\boldsymbol{y}/\sigma_\epsilon^2)\right) \end{aligned}$$

where

$$\mathbf{A} = \left(\frac{1}{\sigma_\epsilon^2}\boldsymbol{\Phi}\boldsymbol{\Phi}^T + \boldsymbol{\Sigma}^{-1}\right)^{-1}.$$

We recognize the above expression to be the variable part of a multivariate Gaussian.[1] Since the denominator is simply a normalizing factor, we conclude that the overall posterior distribution is Gaussian as follows:

$$\boldsymbol{w} \mid \boldsymbol{y}, \boldsymbol{\Phi} \sim \mathcal{N}(\frac{1}{\sigma_\epsilon^2}\boldsymbol{\Sigma}_{\boldsymbol{w}|\mathcal{D}}\boldsymbol{\Phi}\boldsymbol{y}, \quad \boldsymbol{\Sigma}_{\boldsymbol{w}|\mathcal{D}}) \quad \text{where } \boldsymbol{\Sigma}_{\boldsymbol{w}|\mathcal{D}} := \left(\frac{1}{\sigma_\epsilon^2}\boldsymbol{\Phi}\boldsymbol{\Phi}^T + \boldsymbol{\Sigma}^{-1}\right)^{-1} \tag{2.8}$$

Finally, we obtain the predictive distribution $\boldsymbol{f}_*(\boldsymbol{\Phi}_*) := \boldsymbol{\Phi}_*^T \boldsymbol{w}$ for new inputs $\mathbf{X}_*$ using the posterior dsitribution of $\boldsymbol{w}$ (2.8). We notice that $\boldsymbol{f}_*(\boldsymbol{\Phi}_*)$ is a linear transformation of the multivariate Gaussian $\boldsymbol{w}$, and thus is a multivariate Gaussian as specified in (2.2):

$$\boldsymbol{f}_*(\boldsymbol{\Phi}_*) \sim \mathcal{N}(\frac{1}{\sigma_\epsilon^2}\boldsymbol{\Phi}_*^T \boldsymbol{\Sigma}_{\boldsymbol{w}|\mathcal{D}}\boldsymbol{\Phi}\boldsymbol{y}, \quad \boldsymbol{\Phi}_*^T \boldsymbol{\Sigma}_{\boldsymbol{w}|\mathcal{D}}\boldsymbol{\Phi}_*) \tag{2.9}$$

2.1.3 Kernel trick

Calculating the expressions in (2.9) requires inverting $d \times d$ matrices where d is the dimension of the feature space. When the feature space is very large, this can become infeasible. However, if we do not work with the covariance matrix $\boldsymbol{\Sigma}$ explicitly, it is possibble to invert an $N \times N$ matrix instead where N is the number of datapoints.

Towards that end, we first unpack the expression in (2.9) by writing $\boldsymbol{\Sigma}_{\boldsymbol{w}|\mathcal{D}}$ in a different form.

Using the Woodbury matrix identity [55] (A.1), we obtain

$$\boldsymbol{\Sigma}_{\boldsymbol{w}|\mathcal{D}} = \left(\frac{1}{\sigma_\epsilon^2}\boldsymbol{\Phi}\boldsymbol{\Phi}^T + \boldsymbol{\Sigma}^{-1}\right)^{-1} = \boldsymbol{\Sigma} - \boldsymbol{\Sigma}\boldsymbol{\Phi}(\sigma_\epsilon^2 \mathbf{I} + \boldsymbol{\Phi}^T \boldsymbol{\Sigma}\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^T \boldsymbol{\Sigma}. \tag{2.10}$$

[1] Notice that $\boldsymbol{\Sigma}^{-1}$ is positive definite if $\boldsymbol{\Sigma}$ is, and $\boldsymbol{\Phi}\boldsymbol{\Phi}^T$ is positive semidefinite. Therefore $\mathbf{A}$ is positive definite.

Substituting (2.10) into (2.9), we see that $\boldsymbol{\Sigma}$ only appears in the predictive mean and covariance in the forms of $\boldsymbol{\Phi\Sigma\Phi}$, $\boldsymbol{\Phi\Sigma\Phi}_*$, $\boldsymbol{\Phi}_*\boldsymbol{\Sigma\Phi}$, and $\boldsymbol{\Phi}_*\boldsymbol{\Sigma\Phi}_*$. Let us define a function $k : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ such that

$$k(\boldsymbol{x}_1, \boldsymbol{x}_2) = \boldsymbol{\phi}(\boldsymbol{x}_1)^T \boldsymbol{\Sigma} \boldsymbol{\phi}(\boldsymbol{x}_2). \tag{2.11}$$

we can derive that k has an inner product form explicitly from the feature map ϕ and the square root (see Appendix A) of prior covariance $\boldsymbol{\Sigma}$:

$$k(\boldsymbol{x}_1, \boldsymbol{x}_2) = \boldsymbol{\psi}(\boldsymbol{x}_1)^T \boldsymbol{\psi}(\boldsymbol{x}_2), \quad \text{where } \boldsymbol{\psi}(\boldsymbol{x}) := \boldsymbol{\Sigma}^{1/2}\boldsymbol{\phi}(\boldsymbol{x}). \tag{2.12}$$

We also introduce the shorthand

$$\mathbf{K}_{\mathbf{X},\mathbf{X}'} := \begin{bmatrix} k(\boldsymbol{x}_1, \boldsymbol{x}'_1) & \cdots & K(\boldsymbol{x}_1, \boldsymbol{x}'_{N'}) \\ \vdots & \ddots & \vdots \\ k(\boldsymbol{x}_N, \boldsymbol{x}'_1) & \cdots & K(\boldsymbol{x}_N, \boldsymbol{x}'_{N'}) \end{bmatrix}$$

where

$$\mathbf{X} = (\boldsymbol{x}_1, \dots, \boldsymbol{x}_N), \quad \mathbf{X}' = (\boldsymbol{x}'_1, \dots, \boldsymbol{x}'_{N'}).$$

Then, we can rewrite the predictive mean and covariance (2.9) as follows:

$$\begin{aligned}
\frac{1}{\sigma_\epsilon^2}\boldsymbol{\Sigma}_{\boldsymbol{w}|\mathcal{D}}\boldsymbol{\Phi y} &= \frac{1}{\sigma_\epsilon^2}\left[\mathbf{K}_{\mathbf{X}_*,\mathbf{X}} - \mathbf{K}_{\mathbf{X}_*,\mathbf{X}}(\sigma_\epsilon^2\mathbf{I} + \mathbf{K}_{\mathbf{X},\mathbf{X}})^{-1}\mathbf{K}_{\mathbf{X},\mathbf{X}}\right]\boldsymbol{y} \\
&= \frac{1}{\sigma_\epsilon^2}\mathbf{K}_{\mathbf{X}_*,\mathbf{X}}[\mathbf{I} - (\sigma_\epsilon^2\mathbf{I} + \mathbf{K}_{\mathbf{X},\mathbf{X}})^{-1}\mathbf{K}_{\mathbf{X},\mathbf{X}}]\boldsymbol{y} \\
&\qquad\qquad \text{since } \mathbf{I} = (\sigma_\epsilon^2\mathbf{I} + \mathbf{K}_{\mathbf{X},\mathbf{X}})^{-1}(\sigma_\epsilon^2\mathbf{I} + \mathbf{K}_{\mathbf{X},\mathbf{X}}) \\
&\qquad\qquad\qquad = (\sigma_\epsilon^2\mathbf{I} + \mathbf{K}_{\mathbf{X},\mathbf{X}})^{-1}\mathbf{K}_{\mathbf{X},\mathbf{X}} + \sigma_\epsilon^2(\sigma_\epsilon^2\mathbf{I} + \mathbf{K}_{\mathbf{X},\mathbf{X}})^{-1} \\
&= \mathbf{K}_{\mathbf{X}_*,\mathbf{X}}(\sigma_\epsilon^2\mathbf{I} + \mathbf{K}_{\mathbf{X},\mathbf{X}})^{-1}\boldsymbol{y}, \qquad (2.13) \\
\boldsymbol{\Phi}_*^T\boldsymbol{\Sigma}_{\boldsymbol{w}|\mathcal{D}}\boldsymbol{\Phi}_* &= \mathbf{K}_{\mathbf{X}_*,\mathbf{X}_*} - \mathbf{K}_{\mathbf{X}_*,\mathbf{X}}(\sigma_\epsilon^2\mathbf{I} + \mathbf{K}_{\mathbf{X},\mathbf{X}})^{-1}\mathbf{K}_{\mathbf{X},\mathbf{X}_*}. \qquad (2.14)
\end{aligned}$$

Instead of calculating k as in (2.12), we can specify such a function k, called a *kernel*, directly without explicitly defining the feature map ϕ and the prior covariance matrix $\boldsymbol{\Sigma}$. A function k is a valid kernel, meaning that there can be implicit feature map and prior covariance matrix associated with it, if and only if for any choice of N and $\mathcal{D} = (\mathbf{X}, \boldsymbol{y})$, the $N \times N$ matrix $\mathbf{K}_{\mathbf{X},\mathbf{X}}$ is positive semidefinite [38].

When the kernel k is a known function that can be evaluated with negligible computational cost, computing the predictive mean and covariance for a Bayesian linear regression problem (2.13) (2.14) is independent of the dimension d of the feature space. This allows us to work with feature spaces with large or even infinite dimensions implicitly, while only having a computational cost dependent on the number of datapoints. This general technique is called the *kernel trick*.

For example, one of the most commonly used kernels, the squared exponential kernel

$$k(\boldsymbol{x}, \boldsymbol{x}') := \exp\left(-\frac{1}{2}\|\boldsymbol{x} - \boldsymbol{x}'\|^2\right)$$

corresponds to a feature space of infinite dimensions [38].

2.1.4 Function space view

We can arrive at the same expression for predictive distribution (2.13)(2.14) from a different problem setup than Bayesian linear regression. Instead of learning the posterior distribution for the parameters $\boldsymbol{w}$ in (2.5) from known datapoints, we can treat the target function f as a (possibly infinite) collection of random variables and obtain a distribution for unknown datapoints conditioned on known datapoins. The former is often called the *weight space view* and the latter the *function space view*. We now introduce Gaussian processes to formalize the function space view.

A random function $f : \mathcal{X} \to \mathbb{R}$ is called a *Gaussian process* if, for every finite set of points $\boldsymbol{x} \in \mathcal{X}^N$, the finite-dimensional random vector $f(\boldsymbol{x})$ is multivariate Gaussian. When $\mathcal{X}$ is finite, as is the case in our consideration for Gaussian processes on graphs, such a Gaussian process is equivalent to a multivariate Gaussian random vector. However, for consistency with prior works, we use the language of Gaussian processes.

Each Gaussian process is uniquely defined by its mean function $m(\cdot)$ and a positive semi-definite covariance kernel $k(\cdot, \cdot')$

$$m(\cdot) = \mathbb{E}(f(\cdot)) \tag{2.15}$$
$$k(\cdot, \cdot') = \mathrm{Cov}(f(\cdot), f(\cdot')) = \mathbb{E}[(f(\cdot) - m(\cdot))(f(\cdot') - m(\cdot'))], \tag{2.16}$$

and we say that

$$f \sim \mathcal{GP}(m, k).$$

Commonly in practice, the training data is preprocessed to be zero-centered. Thus we also take $m(\cdot)$ to be zero. A more general treatment can be found in [38].

To adopt the same notation as for the weight space view, we consider the case when $\mathcal{X} = \mathbb{R}^n$. However, in general $\mathcal{X}$ can take the form of any input space and our analysis remains the same except for notation.

Given a set of data $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$ from f where $\boldsymbol{x}_i \in \mathbb{R}^n$ is the input and

$$y_i = f(\boldsymbol{x}_i) + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma_\epsilon)$$

is the observation with noise, we can simply make use of the conditional property (2.4) of multivariate Gaussian to incorporate $\mathcal{D}$ into predictive distributions $\boldsymbol{f}_*$ of f at new inputs $\mathbf{X}_*$. For that, we need to write down the joint Gaussian distribution of $\boldsymbol{y}$ and $\boldsymbol{f}_*$:

$$\begin{bmatrix} \boldsymbol{y} = f(\mathbf{X}) + \boldsymbol{\epsilon} \\ \boldsymbol{f}_* = f(\mathbf{X}_*) \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \mathbf{K}_{\mathbf{X},\mathbf{X}} + \sigma_\epsilon \mathbf{I} & \mathbf{K}_{\mathbf{X},\mathbf{X}_*} \\ \mathbf{K}_{\mathbf{X}_*,\mathbf{X}} & \mathbf{K}_{\mathbf{X}_*,\mathbf{X}_*} \end{bmatrix}\right). \tag{2.17}$$

Then, Equation (2.4) gives that

$$\boldsymbol{f}_* \mid \boldsymbol{y} \sim \mathcal{N}(\mathbf{K}_{\mathbf{X}_*,\mathbf{X}}(\mathbf{K}_{\mathbf{X},\mathbf{X}} + \sigma_\epsilon \mathbf{I})^{-1}\boldsymbol{y}, \quad \mathbf{K}_{\mathbf{X}_*,\mathbf{X}_*} - \mathbf{K}_{\mathbf{X}_*,\mathbf{X}}(\mathbf{K}_{\mathbf{X},\mathbf{X}} + \sigma_\epsilon^2 \mathbf{I})^{-1}\mathbf{K}_{\mathbf{X},\mathbf{X}_*}). \tag{2.18}$$

The expressions above coincide with those derived from the weight space view (2.13) (2.14). In fact, for every Bayesian linear regression problem, there is a corresponding Gaussian process with kernel calculated from the feature map and prior covariance (2.12). On the other hand, every kernel has a corresponding (possibly infinite-dimensional) feature map according to Mercer's theorem [38], thus every Gaussian process has a corresponding Bayesian lienar regression problem.

From the function space view, it is more apparent that the key ingredient in defining such a regression model is the covariance function k. Commonly used kernels often correspond to infinite-dimensional feature spaces, making the kernel approach more powerful than Bayesian linear regression models with explicit feature space and prior covariance. Kernel design is therefore a major area of research for Gaussian process methods.

2.2 Kernels on graphs

Let $G = (V, E, w)$ be a graph with node set V, edge set $E \in V \times V$, and weight function $w : E \to \mathbb{R}$ on the edges. We assume for now that G is undirected, and we will discuss directedness in Section 3.3.

In this work, we consider the setting where the input space for Gaussian processes is the node set V of the given graph. In the notation of Section 2.1.4, the input space $\mathcal{X}$

corresponds to the set of nodes in G, which we usually identify as $\{1, 2, \ldots, |V|\}$. For simplicity of notation, we sometimes write the associated finite-dimensional vectors of the target function f as $\boldsymbol{f} \in \mathbb{R}^{|V|}$—that is, $f(i) = (\boldsymbol{f})_i$ for node $i \in V$.

From a prior design perspective, the key design choice for kernels on weighted undirected graphs lies in how to translate the topology of the graphs, as specified by edges and weights, into a positive semi-definite function between nodes. Most approaches for achieving this [22, 39] are either defined from or can be shown to rely on the graph Laplacian, defined as $\mathbf{\Delta} = \mathbf{D} - \mathbf{A}$ where $\mathbf{D}$ is the diagonal degree matrix and $\mathbf{A}$ is the adjecency matrix, to capture the strength of the connections between nodes [3]. Some examples ([3]) include

- the graph exponential kernel, defined as

$$k_\Psi(i, j) = \sum_{n=1}^{N} \Psi(\lambda_n) \boldsymbol{f}_n \boldsymbol{f}_n^T \quad \text{with } \Psi(\lambda) = \alpha e^{-\frac{\kappa^2}{4}\lambda},$$

- the graph matérn kernel, defined as

$$k_\Psi(i, j) = \sum_{n=1}^{N} \Psi(\lambda_n) \boldsymbol{f}_n \boldsymbol{f}_n^T \quad \text{with } \Psi(\lambda) = \alpha \left(\frac{2\nu}{\kappa^2} + \lambda \right)^{-\frac{\nu}{2}},$$

- the random walk kernel, defined as

$$(\mathbf{I} - (1 - \alpha)\mathbf{D}^{-1/2} \mathbf{\Delta} \mathbf{D}^{-1/2})^p.$$

In the above, $(\lambda_n, \boldsymbol{f}_n)$ are eigenvalues and eigenvectors of the graph Laplacian $\mathbf{\Delta}$, and α, κ, ν, p are hyperparameters that can be chosen by, for example, maximizing the marginal likelihood [38].

A Gaussian process prior (before conditioning on data) built using such Laplacian-based kernels represents a random function $f : V \to \mathbb{R}$ defined on the nodes of the graph, which reflects the structure and connectivity of G by ensuring that the values of nodes are more similar when the connections between them are stronger. We will use such kernels both as baselines, and as potential building blocks for kernels that incorporate additional structure in the rest of the sections.

Before proceeding, we note that these ideas admit a number of extensions and variations that differ from the ones we study. For example, [34] study graphs with node features, [57, 50] study multi-output processes, and [52, 23] study Gaussian processes $f : \mathcal{G} \to \mathbb{R}$ whose input is an actual graph $G \in \mathcal{G}$ within some space of graphs $\mathcal{G}$. We focus on the case where the unknown quantity is a real-valued function on the nodes of a given graph.

2.3 Interdomain GP and constrained GP

In our construction of the graph kernel, we start with a given covariance matrix and then transform it into a kernel that satisfies the linear dependencies. This falls into the general technique of interdomain Gaussian processes, which was first introduced in the context of sparse Gaussian process inference [26] and has been used in numerous applications to enable computationally efficient Gaussian process models [28, 48, 47]. The interdomain technique has also been used for graph kernel construction in [34].

Our goal of fitting Gaussian processes that satisfy certain linear dependencies falls into the general area of constrained Gaussian processes. There have been many works using various constraint methods for Gaussian processes [45]. In particular, several works [19, 25, 2, 37] in the numerical ODEs and PDEs literature, as well as [10] in the statistics literature, also use transformed covariance matrix for linear constraints.

2.4 Offline reinforcement learning

Reinforcement learning (RL) is a machine learning framework for making sequencial decisions by interacting with the environment. An RL environment can be modeled as a Markov decision process, which includes

- a set of states $\mathcal{S}$,
- a set of actions $\mathcal{A}$,
- a transition model that for all $s \in \mathcal{S}$, $a \in \mathcal{A}$, and $s' \in \mathcal{S}$. specifies

  $$P(s_{t+1} = s' \mid s_t = s,\ a_t = a),$$

  which is the probability of transitioning to state s' at time $t+1$ given that we executed action a in state s at time t,
- a reward function that for all $s \in \mathcal{S}$, $a \in \mathcal{A}$, and $s' \in \mathcal{S}$, specifies $R(s, a, s')$, the reward for transitioning from s to s' by executing action a.

Note that the reward function can be as simple as a deterministic function dependent only on s, as in most maze environments, but it can also be a stochastic function depending on s, a', and s'.

Given the environment, our goal is to learn a *policy* π to (deterministically or stochastically) select an action given the current state $\mathcal{S}$. A policy π_* is optimal when it maximizes the accumulated reward over all time steps

$$\pi^* = \text{argmax}_\pi \sum_{t=0}^{h} \gamma^t \, \mathbb{E}_\pi(r_t).$$

Here r_t is the reward obtained in step t, and h is the total number of time steps, also called the *horizon*. There is also a fixed constant γ, called the *discount factor*, with $0 \leq \gamma \leq 1$. The discount factor controls how important future time steps are with respect to a current time step, and a discount factor strictly less than 1 is required to learn a sensible policy when the horizon h is infinite.

When the MDP is known, we can compute the optimal policy exactly via dynamic programming [44]. In RL, however, the transition model and reward function for the environment are a priori unknown, and information about them can only be gained by interacting with the environment and observing the transition and reward for each executed time step. The goal is to learn a good policy given interactions with the environment, called *trajectories*, in the form of state-action-state triples $\{(s_i, a_i, s_{i+1})\}_{i=1}^{h-1}$.

The setup where the executed RL algorithm, also called an *agent*, obtain trajectories by interacting with the environment in real time is called *online* RL. On the other hand, the setup where the agent learns from pre-recorded trajectories is called *offline* RL. Online RL allows more flexible exploration of the environment, whereas offline RL are preferred in safety-critical tasks. In this work, we are concerned with offline RL.

2.5 Reinforcement learning with graph structure

Our application of graph kernels to RL is based on the observation that, the states and actions of a discrete-state RL problem form a natural graph structure, where nodes are states and edges are transitions between states. Learning a policy while taking into account this graph structure might be more efficient than treating the state and action labels as Euclidean data, as for common deep RL models [33, 24]. There are numerous existing works that also incorporate graph structure into RL algorithms. [30] and [51] use the state-action graph in maze environments to obtain low-dimensional vector embeddings of the states, and use these embeddings as state features that feed into Q-networks. They show improved performance over using raw state labels. [11] solves the state-space coverage problem using RL and GNN, and uses the state-action graph to faciliate exploration and transferring to unseen state graphs.

Some other works use graph structures existing in the environment that do not use states themselves as nodes, and optimization can be done as message passing on these graphs. For example, [53] models the body parts of a robot as a graph; [18] models a multi-agent environment as a graph; [9] uses the graph structure in dialogues for dialogue policy optimization; [49] uses a causal relation graph for probabilisitc planning.

Chapter 3

Graph Gaussian processes with linear dependencies

In this chapter, we introduce our method, analyze its properties, and demonstrate its efficiency and robustness on synthetic data.

3.1 Graph kernels with linear dependencies up to noise

We now describe how to incorporate linear dependencies into the covariance kernels used to define Gaussian processes on graphs. The basic idea is to start with a graph kernel that encodes assumptions about the noise terms on the graph, and then transform it into a kernel that obeys the linear dependencies. In what follows, we begin by outlining the precise construction, and then explore the prior information it introduces in more detail and explore its connections with other kernels.

3.1.1 Kernel construction

Let $G = (V, E, w)$ be a graph. We consider the noisy version of the constraints (1.1):

$$f(i) = \delta(i) + \sum_{j \in V, (i,j) \in E} m_{ij} f(j). \tag{3.1}$$

for all $i \in V$. Here $\delta(i)$ is the noise term at node i.

We seek to construct a Gaussian process prior for the target function f in (3.1). We overload notation and still write f for the Gaussian process. We assume that the noise terms are represented by a Gaussian process $\delta : V \to \mathbb{R}$ whose distribution is $\delta \sim \mathcal{GP}(\mathbf{0}, k_\delta)$ on graph G. For example, the simplest choice can be to take δ to be i.i.d. at each node.

Then, Equation (3.1) becomes an equation between two graph Gaussian processes, namely f and δ, which are related to each other via an interdomain operator representing the linear transformation defined by the dependencies. To see this explicitly, we can reinterpret this equation using matrices and vectors, and write $\boldsymbol{f} = \boldsymbol{\delta} + \mathbf{M}\boldsymbol{f}$, which rearranges into

$$\boldsymbol{f} = (\mathbf{I} - \mathbf{M})^{-1}\boldsymbol{\delta} \tag{3.2}$$

This equation therefore defines a Gaussian random vector at every node, and hence a graph Gaussian process $f : G \to \mathbb{R}$, whose covariance kernel we denote by k. Since k is the covariance kernel of a linear transformation of a Gaussian random vector, it follows immediately from (2.4) that the resulting kernel is positive semi-definite, and has the form

$$k(i,j) = [(\mathbf{I} - \mathbf{M})^{-1}\boldsymbol{\Lambda}(\mathbf{I} - \mathbf{M})^{-T}]_{ij} \tag{3.3}$$

where the matrix $\boldsymbol{\Lambda}$ is defined as $\Lambda_{ij} = k_\delta(i,j)$. We call the kernel constructed this way a *graph kernel with linear dependencies* $\mathbf{M}$.

3.1.2 Understanding the prior information introduced by linear dependencies

We now investigate qualitatively how our kernel incorporates the structure of the linear dependencies.

We focus on differences with the most-common class of graph kernels, namely Laplacian-based kernels, as introduced in Section 2.2. Though they are in general not distance-based, most Laplacian-based kernels propagate uncertainty in a manner that resembles Euclidean stationary kernels. One way to see this is through their connection with random walks: for instance, certain forms of the graph diffusion kernel can be shown to equal the unnormalized density of a random walk on the graph. The correlations induced by such kernels will tend to be large in regions of the graph that are close-by, and small in regions that are far away.

The linearly constrained kernels of Section 3.1 do not generally behave this way. One key difference is *directedness*: Laplacian-based graph kernels are only defined on weighted

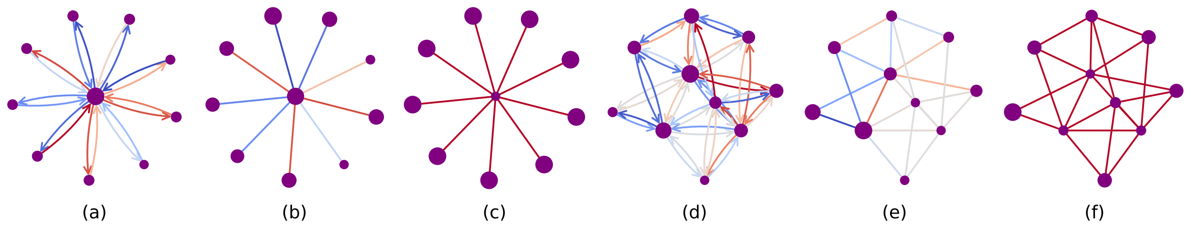

Figure 3.1: Visualization of a linearly constrained kernel on a star graph (first 3 graphs), and a random graph (last 3 graph). (a) and (d): linear dependencies on the graph. Each edge is shown as red if the coefficient is positive, and blue if it is negative. Less-transparent edges correspond to coefficients that are greater in absolute value. (b) and (e): values of a linearly constrained kernel (lck), where the size of each node indicates the kernel's prior variance, and color/transparency indicates the sign and magnitude of the kernel's value between the respective edges. (c) and (f): values of a graph Matérn kernel with smoothness $\nu = 3/2$, for comparison.

undirected graphs, whereas our model allows directed linear dependencies, i.e. linear dependencies where the coefficient matrix $\mathbf{M}$ is asymmetric. This allows our kernel to vary the degree of correlation according to the directionality of the dependencies. In particular, one can use this extra flexibility to construct kernels on directed graphs, which we explore further in Section 3.3. Furthermore, Laplacian-based graph kernels have non-negative prior covariance, whereas our kernel can model *negative correlations* by using negative coefficients in linear constraints.

In Figure 3.1 and Figure 3.2, we visualize the properties discussed above on two graphs with 10 nodes, namely a star graph and a random graph. In Figure 3.1, one can see that the covariances represented by our kernel reflect the structure of the dependencies: in particular, stronger dependencies tend to result in stronger covariances. The kernel tends to assign negative correlations to nodes whose corresponding linear dependencies are negative. In cases where the linear dependencies are positive in one direction, but negative in the opposite direction, both positive and negative correlations can result, depending on the strength of the dependencies and overall structure of the graph. In Figure 3.2, one can see that the signedness pattern of random draws from a GP prior with linearly constrained kernel is similar to the signedness pattern of the ground truth function. In comparison, random draws from the graph Matérn kernel have values with the same signedness, or values that vary gradually based on the geometry of the graph.

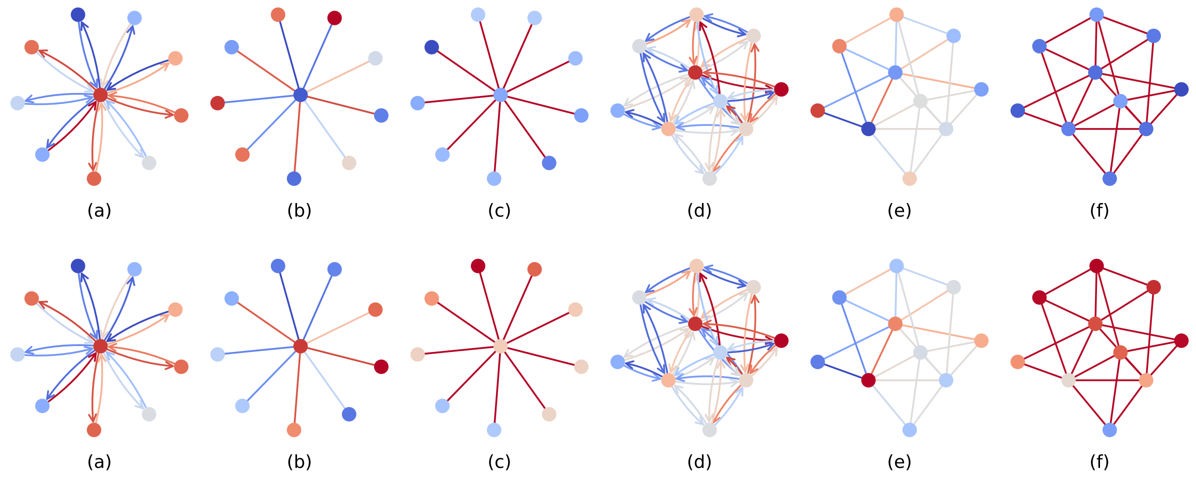

Figure 3.2: Visualization of random draws from a linearly constrained kernel on a star graph (first 3 graphs), and a random graph (last 3 graph). First row is a draw with seed 1. Second row is a draw with seed 2. The edge colors have the same meaning as in Figure 3.1. (a) and (d): ground truth function on the graph. (b) and (e): values of a function randomly drawn from GP prior with linearly constrained kernel. (c) and (f): values of a function randomly drawn from GP prior with graph Matérn kernel with smoothness $\nu = 3/2$, for comparison. A node is shown as red if the function value is positive, and blue if it is negative. Less-transparent nodes correspond to function values of greater absolute value.

3.2 Robustness of GP prediction with linearly constrained kernels

Our construction requires prior information on the matrix $\mathbf{M}$ that specifies linear dependencies between nodes on graph G. However, accurate knowledge of $\mathbf{M}$ cannot be guaranteed in most applications. In this section, we explore theoretically the robustness of our model in the case that the linear dependencies in $\mathbf{M}$ are misspecified. In Section 3.4, we evaluate the robustness of our model empirically on synthetic data.

3.2.1 Tolerance for misspecified linear dependency

In this section, we apply known results from [14] to give a probabilistic bound for how much the posterior GP deviates from the ground truth function when using a linearly constrained kernel with misspecified linear dependencies. The result in [14] uses the Reproducing Kernel Hilbert Space (RKHS) framework, for which we refer the reader to [20] for an introduction.

Theorem 1 ([14]) *Let $D \neq \emptyset$ be a set and $k : D \times D \to \mathbb{R}$ a positive definite kernel with corresponding RKHS $(H_k, \|\cdot\|_k)$. We define the kernel matrix $\mathbf{K}_N = (k(x_i, x_j))_{i,j=1,\dots,N}$ and the column vectors $\boldsymbol{k}_N(x) = (k(x_i, x))_{i=1,\dots,N}$ and $\boldsymbol{y}_N = (y_i)_{i=1\dots,N}$.*

Let f be a function such that $f \in H_k$ with $\|f\|_k \leq B$ for some $B \geq 0$. Let $x_1, \dots, x_N \in D$ be given and $\epsilon_1, \dots, \epsilon_N$ be real-valued independent Gaussian random variables with variance σ_ϵ^2. Furthermore, define $y_n = f(x_n) + \epsilon_n$ for all $n = 1, \dots, N$.

Consider a Gaussian process $g \sim \mathcal{GP}(\mathbf{0}, k)$ and denote its posterior mean function by μ_N, its posterior covariance function by k_N and its posterior variance by $\sigma_N^2(x) := k_N(x, x)$, w.r.t. to data $(x_1, y_1), \dots, (x_N, y_N)$, assuming in the likelihood independent Gaussian noise with mean zero and variance $\lambda \geq 0$. Then for any $\delta_p \in (0, 1)$ with

$$\eta_N(x) = \sigma_\epsilon \|(\mathbf{K}_N + \lambda \mathbf{I}_N)^{-1} \boldsymbol{k}_N(x)\| \sqrt{N + 2\sqrt{N}\sqrt{\log(1/\delta_p)} + 2\log(1/\delta_p)} \tag{3.4}$$

one has

$$P\left[|\mu_N(x) - f(x)| \leq B\sigma_N(x) + \eta_N(x)\, \forall x \in D\right] \geq 1 - \delta_p. \tag{3.5}$$

To apply Theorem 1, we also need the following theorem from [40]:

Theorem 2 *Denote by* $\mathbf{P} \in \mathbb{R}^{m\times m}$ *a positive semidefinite matrix and denote by* $\mathcal{H}$ *the image of* $\mathbb{R}^m$ *under* $\mathbf{P}$*. Then* $\mathcal{H}$ *with dot product* $\langle f, f\rangle_{\mathcal{H}} := \langle \boldsymbol{f}, \mathbf{P}\boldsymbol{f}\rangle$ *is a RKHS and its kernel is* $k(i,j) = [\mathbf{P}^{-1}]_{ij}$ *where* $\mathbf{P}^{-1}$ *denotes the pseudo-inverse if* $\mathbf{P}$ *is not invertible.*

Now, let $\mathbf{M}$ be the true linear dependencies that generated the target function $\boldsymbol{f}$ in (3.2). Let $\mathbf{M}'$ be the linear dependencies assumed in the linearly constrained kernel k in (3.3), which we can select such that $\mathbf{I} - \mathbf{M}'$ is invertible. Then k has the matrix form $\mathbf{K} := (\mathbf{I} - \mathbf{M}')^{-1}\boldsymbol{\Lambda}(\mathbf{I} - \mathbf{M}')^{-T}$. According to Theorem 2, the kernel k has the corresponding RKHS $(H, \|\cdot\|_k)$ that is the image of $\mathbf{K}^{-1}$ with norm $\|\cdot\|_k = \langle \cdot, \mathbf{K}^{-1}\cdot\rangle$.

Then, we can apply Theorem 1 to k and f. In the expressions (3.4) and (3.5), the variance for observational noise σ_ϵ can be specified based on the task at hand, and the value of λ is a modeling choice. The rest of the quantities are known. It therefore remains to upperbound $\|f\|_k$ by a constant B.

In our case, since $\boldsymbol{f} = (\mathbf{I} - \mathbf{M})^{-1}\boldsymbol{b}$ for some random vector $\boldsymbol{b}$ that models local noise, we can calculate $\|f\|_k$ as

$$\|f\|_k = \boldsymbol{f}^T\mathbf{K}^{-1}\boldsymbol{f} = \boldsymbol{b}^T(\mathbf{I} - \mathbf{M})^{-T}(\mathbf{I} - \mathbf{M}')^T\boldsymbol{\Sigma}^{-1}(\mathbf{I} - \mathbf{M}')(\mathbf{I} - \mathbf{M})^{-1}\boldsymbol{b}. \tag{3.6}$$

An upperbound on the expression above can be obtained in different ways, depending on the modeling for the task. For example, the norm of $\boldsymbol{b}$ and the magnitude of entries of $\mathbf{M}$ can be specified by domain knowledge, and the form of $\boldsymbol{\Sigma}$ can be chosen to ease computation (e.g. $\boldsymbol{\Sigma}$ can be a scalar multiple of the identity matrix as in our experiments). Matrix perturbation bounds such as those in [42] can also be used to bound the norm of $(\mathbf{I} - \mathbf{M}')(\mathbf{I} - \mathbf{M})^{-1}$ if how much $\mathbf{M}'$ deviates from $\mathbf{M}$ is easy to model.

3.2.2 Graph Matérn kernels as linearly constrained kernels

In the previous section, we showed that when the linear dependencies are not exactly accurate but close to the true dependencies, the GP model with our kernel still gives predictions that are close to the target function. However, there might be applications where information about the linear dependencies is too inaccurate for Theorem 1 to be meaningful, or no accurate information about the dependencies is available at all. In these cases, we show that our kernel can fall back to Laplacian-based kernels.

Indeed, with appropriate modeling choices, graph Matérn kernels are linearly constrained kernels. Let the noise covariance matrix $\boldsymbol{\Lambda} = c_1\mathbf{I}$ be a constant multiple of the identity. Let $\mathcal{A} = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$ be the normalized adjacency matrix of G, and let $\mathbf{\Delta} = \mathbf{I} - \mathcal{A}$ be the symmetric normalized graph Laplacian. Let the constraint matrix be

$$\mathbf{M} = (1 - c_2)\mathbf{I} - \mathbf{\Delta}. \tag{3.7}$$

Using this choice, we obtain

$$k(i, j) = [c_1(\mathbf{I} - \mathbf{M})^{-2}]_{ij} = [c_1(c_2\mathbf{I} + \mathbf{\Delta})^{-2}]_{ij}$$

which is a graph Matérn kernel with parameters $\alpha = c_1$, $\nu = 4$, and $\kappa = \sqrt{8/c_2}$. As a consequence, such choices of $\mathbf{M}$ will result in similar prior information and performance as graph Matérn kernels. This connection allows one to specify linearly constrained kernels which mirror the behavior of Laplacian-based kernels where needed.

3.3 Kernels for directed graphs via linear dependencies

Since Laplacians arising from asymmetric adjacency matrices are not positive semi-definite, Laplacian-based kernels apply only to undirected graphs. Hence, there are less kernels available for directed graphs than for undirected graphs, and many examples are constructed by first converting the adjacency or Laplacian matrix into suitable analogs which resemble undirected graphs—see for example [29], which suggest using a graph advection operator for this purpose.

One can use linearly constrained kernels to define a kernel for directed graphs. For a weighted directed graph G, we assume we have a well-defined base kernel, such as the identity kernel $k(i, j) = \mathbb{1}_{i=j}$. By choosing $\mathbf{M} = \mathbf{A}$ to be the (asymmetric) adjacency matrix of G, we obtain a linearly constrained kernel which reflects the structure of the directed graph.

For this construction to be well-defined, the matrix $\mathbf{I}-\mathbf{A}$ needs to either be invertible, or replaced by a suitable analog such as a pseudoinverse. In general, there is no known natural graph-theoretic characterization for the invertibility of adjacency matrices. However, in all our synthetic and applied experiments, the graphs encountered satisfied this requirement.

There are many further design choices in constructing such a kernel for directed graphs, based on the application at hand. For example, one can take into account the in-degree, out-degree, or both when choosing the base kernel, and one can also use the degrees to normalize the adjacency matrix before using it to construct kernels.

3.4 Experiments on synthetic graphs with linear dependencies

We now empirically evaluate the accuracy and data-efficiency of GP model with linearly constrained kernels. We also demonstrate the robustness of model performance against misspecified linear dependencies, and explore how the choice of the base kernel for the noise terms affects model performance.

Given a graph G with n nodes, we generate an $n \times n$ coefficient matrix $\mathbf{M}$ by replacing each non-zero entry of the adjacency matrix A of G with a random number sampled uniformly from $[-1, 1]$. We then define a synthetic function on the nodes of the graph, through the vector representation $\boldsymbol{f} = (\mathbf{I} - \mathbf{M})^{-1}\boldsymbol{\zeta}$ where $\boldsymbol{\zeta} \sim \mathrm{N}(\mathbf{0}, \mathbf{I})$. We consider three kinds of random graphs, including a Barabási–Albert graph with 500 nodes and 3 edges per node, an Erdős–Rényi graph with 500 nodes and parameter $p = 0.2$, and a Gaussian random partition graph with 500 nodes. Full experimental details are given in Appendix B.

For each $\boldsymbol{f}$, we evaluate GP predictions for test values of $\boldsymbol{f}$, and examine the mean absolute error with standard error for each model, against the number of training nodes. We consider the graph Matérn kernel, the graph exponential kernel, the random walk kernel, linearly constrained kernel as a Matérn kernel as described in Section 3.2.2, and linearly constrained kernel (lck) with base kernel $\mathbf{I}$, $10\mathbf{I}$, $100\mathbf{I}$, respectively. For each model, we construct a version with exact $\mathbf{M}$, a version with $\mathbf{M}$ perturbed by adding independent Gaussian noise sampled from $\mathrm{N}(0, 0.1)$ to each nonzero entry, and a version similarly perturbed with noise from $\mathrm{N}(0, 0.2)$. Finally, we include a linearly constrained kernel which instead uses a normalized adjacency matrix as in (3.7). To assess variability, we repeat each experiment ten times.

Figure 3.3 gives the results, which show that all kernels that incorporate exact or perturbed $\mathbf{M}$ clearly outperform the two kernels that do not use $\mathbf{M}$. We further see that all versions of our kernel perform comparably when $\mathbf{M}$ is well-specified. On closer inspection, kernels with lower base kernel variance perform better in this case. In misspecified cases, kernels with higher base kernel variance perform better.

These results illustrate that our construction does reflect the structure of the linear dependencies and improves the quality of predictions in cases where this structure is present, as expected. Moreover, linearly constrained kernels are robust against moderate perturbations.

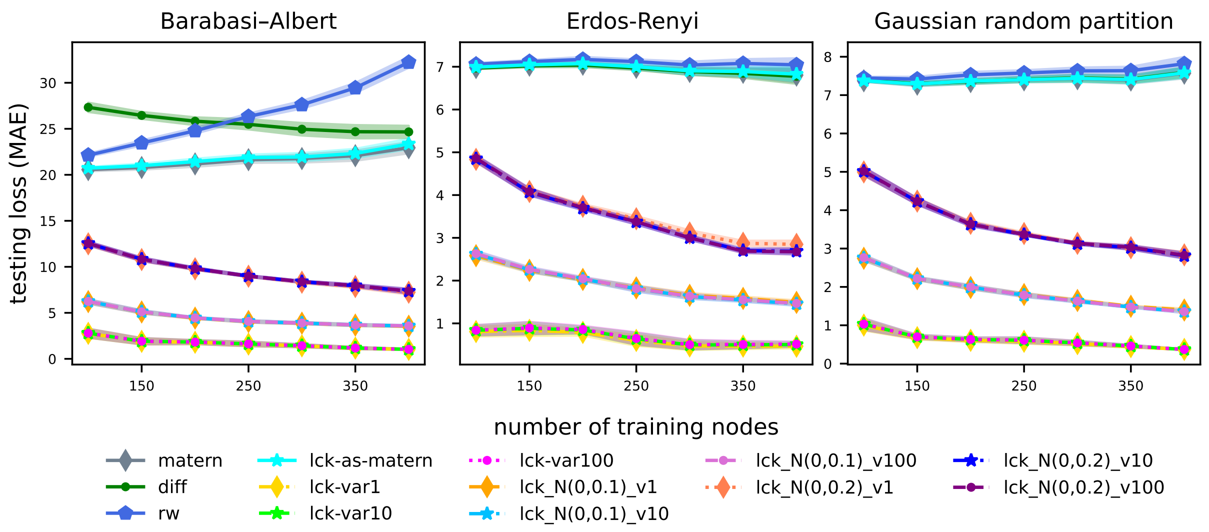

Figure 3.3: Mean absolute error of predictions at test nodes for three different random synthetic graphs, as a function of data size. We compare several variants of linearly constrained kernel and a graph Matérn baseline. As we aim to verify, kernels which incorporate linear constraints significantly outperform the Matérn baseline, even when the linear constraints are perturbed by noise. When using the normalized adjacency matrix as in (3.7) instead of the constraints, our kernel performs similarly to a Matérn kernel. Thus, our model reverts to baseline performance in cases when the linear constraints are misspecified or not available.

Chapter 4

Applications

In this chapter, we demonstrate the usefulness of linearly constrained kernels in practical domains by applying GP models with linearly constrained kernels to two tasks, traffic speed prediction and offline discrete-state reinforcement learning.

4.1 Traffic Speed Prediction

We study the task of traffic speed prediction, a regression problem where the input is speed data on nodes of a graph representing a road network. Given speed sensor data on certain nodes, our task is to predict traffic speed on other nodes in the network. We begin by showing how to derive a set of approximate linear dependencies between speeds on adjacent nodes, and then construct our kernel based on these dependencies.

4.1.1 Model derivation

For each node i, let N_i be the number of cars per lane passing by node i during a time interval of length t, let d_i and s_i be the distance between, resp. speed of, cars around node i. We can model s_i as

$$s_i = \frac{d_i}{t/N_i}. \tag{4.1}$$

Let p_{ji} be the transition probability of a car going from an adjacent node j to i, and let ℓ_i be the number of lanes around node i. We assume that the traffic conditions on different lanes around node i are the same. Based on the conservation of the number of cars on the road—that is, based on the fact that the cars exiting a road has to enter another road —we have that

$$\ell_i \cdot N_i = a_i + \sum_{j\colon\, j\sim i} p_{ji} \cdot \ell_j \cdot N_j \tag{4.2}$$

Here, the term a_i represents cars which appear from roads not included in the graph—for instance, because they are too small to model at the scale in question. Summation in (4.2) is performed over all nodes j such that cars can go from j to i. Note that, in spite of the probabilistic interpretation of p_{ij}, the sum of these values need not necessarily equal 1, as there might be cars turning to roads not included in the graph or going off roads. Combining (4.1) and (4.2) gives

$$\ell_i \frac{s_i}{d_i} = a_i + \sum_{j\colon\, j\sim i} p_{ji} \ell_j \frac{s_j}{d_j}. \tag{4.3}$$

which define the linear dependencies used to construct our kernel. In practice, the relevant terms in (4.3) can be easily collected from traffic sensors. One can also straightforwardly either simplify these assumptions, or extend them to more complex forms that allow for more accurate modeling.

4.1.2 Application and results

Next, we apply the above modeling to a real-world data. We consider a simplified variant of the linear constraints derived in Section 4.1, constructed as follows. Assume that $\ell_i = d_i$ for all nodes, and absorb the scalars into a_i. Take the transition probabilities to be $1/\deg(j)$ where $\deg(j)$ is the number of edges adjacent to j, and introduce a global multiplicative scalar b for the transition probabilities to account for cars going off the modeled network. With these choices, we obtain the linear dependencies

$$s_i = a_i + b \sum_{j\colon\, j\sim i} \frac{1}{\deg(j)} s_j. \tag{4.4}$$

which we use to construct a linearly constrained kernel. Here our target function f maps the node i to the speed at node i, that is, $\boldsymbol{f}_i = s_i$. The recorded speed on the training nodes corresponds to entries of the noisy observation $\boldsymbol{y}$ as in (2.18), and $b/\deg(j)$ corresponds to the coefficients m_{ij} as in (3.3). Additional details on this construction are given in Appendix B.

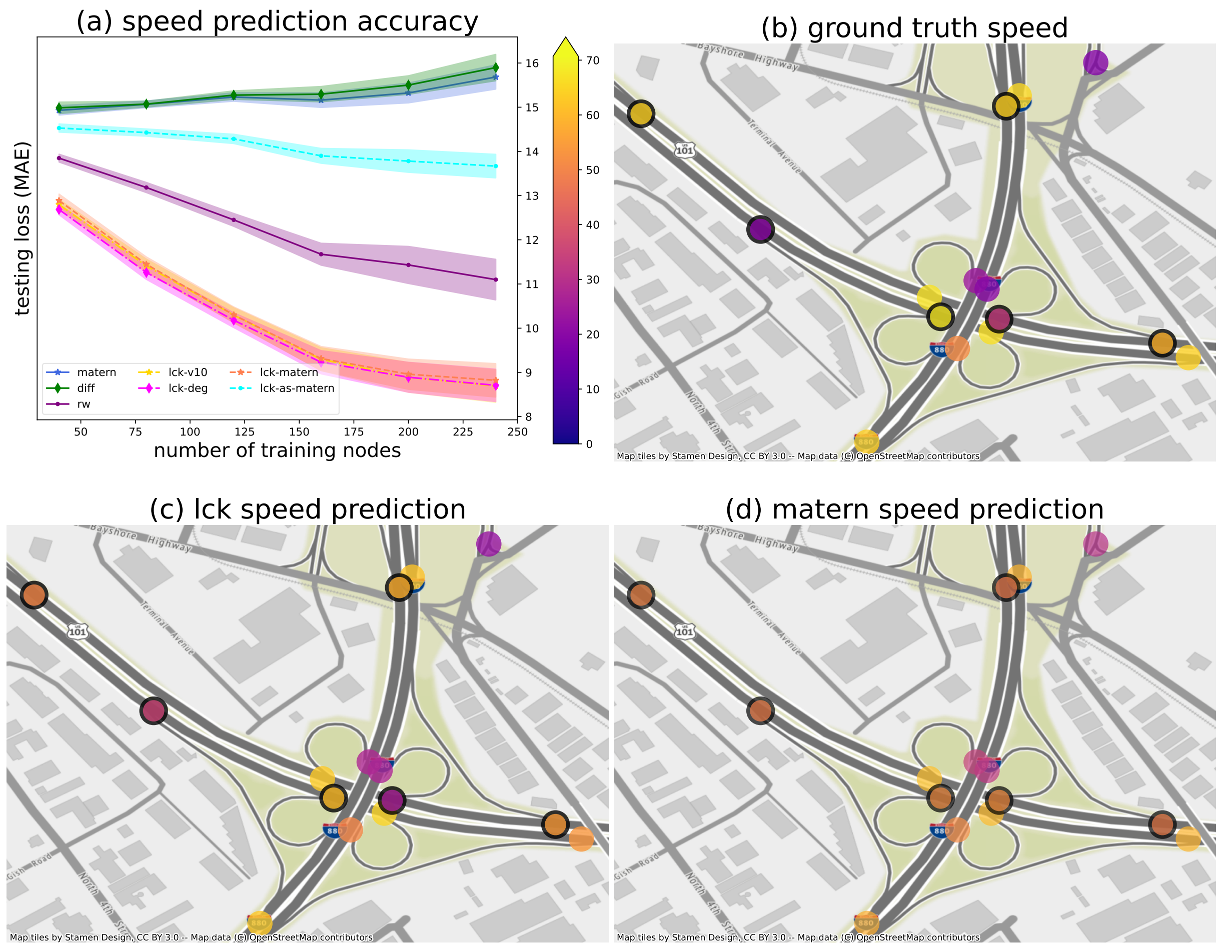

Figure 4.1: (a): Comparison for test loss on traffic data between several variants of linearly constrained kernels and a Matérn baseline. Here, the x-axis is the number of training nodes, and the y-axis is the testing loss. (b): speed sensor data at a highway interchange. (c) and (d): predictions by our kernel, and the Matérn kernel. Nodes without black borders are training nodes, and those with black borders are testing nodes. We see that linearly constrained kernels, which are able to incorporate the directed structure of the road network, achieve significantly lower loss than undirected baselines.

We use this kernel to predict traffic on a highway network. Following [3], we choose a road network located in San Jose, California, using street data from OpenStreetMap [35] and sensor data from the California Performance Measurement System [8]. The resulting graph has 979 nodes and 1138 edges, with speed data available at 283 of the nodes.

Among the set of nodes with sensor data, we train on a randomly selected subset, and then test on the rest of the nodes with sensor data. We include linearly constrained kernels using the same linear dependencies specified in (4.4), with base kernel $10\mathbf{I}$, $10\mathbf{D}$ (diagonal degree matrix), and a trainable graph Matérn kernel, respectively. We include a graph Matérn kernel by itself as a baseline, as well as a graph diffusion kernel and a random walk kernel. We train all models for 1000 iterations to fit hyperparameters, and repeat each experiment ten times.

Figure 4.1, we plot mean absolute test error as a function of the number of training nodes. We see that linearly constrained kernels consistently outperform baselines which do not incorporate constraints. Figure 4.1 further illustrates a zoomed-in view of a highway interchange to visualize the differences in predictions between constraint-based kernels and those which do not incorporate this information. We see that the graph Matérn kernel is less effective at capturing the true speed function in situations when the speeds change sharply between nearby nodes, compared to the linearly constrained kernel shown, which benefits from the additional prior information included.

4.2 Offline Reinforcement Learning

Following the intuition that states and actions naturally form a graph with linear structure in RL problems, we apply GP models with linearly constrained kernel to discrete-state RL.

4.2.1 Linear dependencies in offline RL

Given a discrete-state reinforcement learning problem with finite state space $\mathcal{S}$, action space $\mathcal{A}$, unknown environment transition probabilities $P : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$, unknown reward function $r : \mathcal{S} \to \mathbb{R}$, and discount factor $\gamma \in [0, 1)$, we assume that we have access to a graph $G = (\mathcal{S}, E)$ whose nodes are states, and each directed edge (s, a, s') corresponds to a pair of reachable states (s, s') labeled by some action a, as well as recorded trajectories in the form of (s, a, s') transition tuples. We estimate the transition probabilities as follows:

$$\tilde{P}(s', a|s) \propto \begin{cases} 1 + \#(s, a, s') & (s, a, s') \in E \\ \#(s, a, s') & \text{otherwise.} \end{cases} \tag{4.5}$$

where $\#(s, a, s')$ is the number of visits to s' after executing a in s observed from the trajectories. We can also marginalize a to obtain $\tilde{P}(s'|s) = \sum_a \tilde{P}(s', a|s)$. Given this transition model, we can define a linear system of equations that expresses dependencies between the unknown value (discounted expected sum of rewards) $V(s)$ at each state s:

$$V(s_i) = r(s_i) + \gamma \sum_{j:(s_i,\_,s_j)\in E} \tilde{P}(s_j|s_i)V(s_j) \tag{4.6}$$

Since the given trajectories may have visited only a small fraction of the states, and the rewards of the unreached states are unobserved, the above system of linear equations cannot be used to directly estimate V. However, we can use the equations to set the coefficients $m_{ij} = \gamma\tilde{P}(s_j|s_i)$ in Equation 3.1. This allows us to use our kernel on the graph G to obtain a distribution over V for the unvisited states.

4.2.2 Experiment and results

To illustrate the applicability of linearly constrained kernels in offline reinforcement learning, we experiment with 30×30 mazes adapted from Gym-maze, where Figure 4.5(a) shows an example maze. The mazes are stochastic, with positive and negative rewards (traps) at different cells, and with teleport portals: we describe full details about the mazes and trajectories used for training in Appendix. The task is to navigate from predefined start states (not necessarily visited in the training trajectories) to the bottom right corner.

We assume the agents have access to the structure of the maze, namely the connectivity of the cells. This is natural in navigation tasks, where an agent can observe the visual aspect of the environment, but does not have knowledge of the interactive aspects of the environment. We follow the formulation in Section 4.2, using (4.6) to construct our kernel. Here the target function is the value function $V : \mathcal{S} \to \mathbb{R}$ that maps a state to the expected cumulative reward at that state. We estimate the value at each state visited in the trajectories using Monte Carlo estimation [44], and this corresponds to the noisy samples $\boldsymbol{y}$ of the target function as in (2.18). The coefficients m_{ij} in (3.3) are defined as $\gamma\tilde{P}(s_j|s_i)$ in this case. We then use our constructed kernel to estimate the value function at all states. Finally, we use (4.5) to convert the estimated values into Q-values. We then evaluate the fixed policy induced by these Q-values on mazes.

As baseline, we use a conservative Q-learning (CQL) model [24]—a canonical baseline for offline RL. Since CQL cannot naturally incorporate the prior modeling graph, we use a CQL model equipped with node embeddings of the graph generated by DeepWalk [36] as another baseline (CQL-DW). Specifically, we feed the embeddings of cells as state labels instead of simply the position of the cells. This has been shown to improve performance of deep Q-learning methods in maze environments [31]. We also include GP models with graph Matérn kernel, graph diffusion kernel, and random walk kernel, which incorporate the graph structure but not the exact relations between the value function of neighboring states.

For each randomly constructed maze, we use the upper left (UL) corner, upper right (UR) corner, lower left (LL) corner, and middle (M) as the initial states for evaluations. For each initial state and a number of given trajectories, we record the cumulative reward of each model for five runs. Moreover, to test the robustness of the models, we use trajectories with different optimality for training. Figure 4.2, Figure 4.3, and Figure 4.4[1] show the average cumulative rewards and standard error over five runs when given training trajectories that are increasingly optimal, i.e. the trajectories have on average fewer steps and lower variance. We note that to control the experiment scale, we have capped all models to a maximum of 1000 steps if they have not reached the goal state. This corresponds to a reward of -0.11 if no traps are visited. The plots have a cut-off at -1.5, so models with lower than -1.5 rewards are not shown.

In total, we see that linearly constrained kernel achieve higher rewards and significantly more stable performance compared to other methods. It especially outforms the other methods when the starting state is different from that of the training trajectories, showing that linearly constrained kernel effectively propagates the value function to unseen states via the graph structure and the linear dependency structure. Moreover, it is least affected by the trajectories being suboptimal, supporting our analysis of its robustness in Section 3.2 and Section 3.4.

Figure 4.5(a) visualizes a specific maze and the trajectories for different methods, starting from the upper-right state. The black dots are cells visited by the five training trajectories, the blue dots are cells visited by CQL-DW, the green dots are cells visited by the Matérn model, and the red dots are cells visited by our model (lck). Our model successfully reaches the goal state from a starting state unseen in the trajectories, while both CQL-DW and the Matérn kernel get stuck in different corners and fail to reach the goal state up until

[1]Best viewed in color. Please contact `yueheng.zhang@uwaterloo.ca` for tabular data.

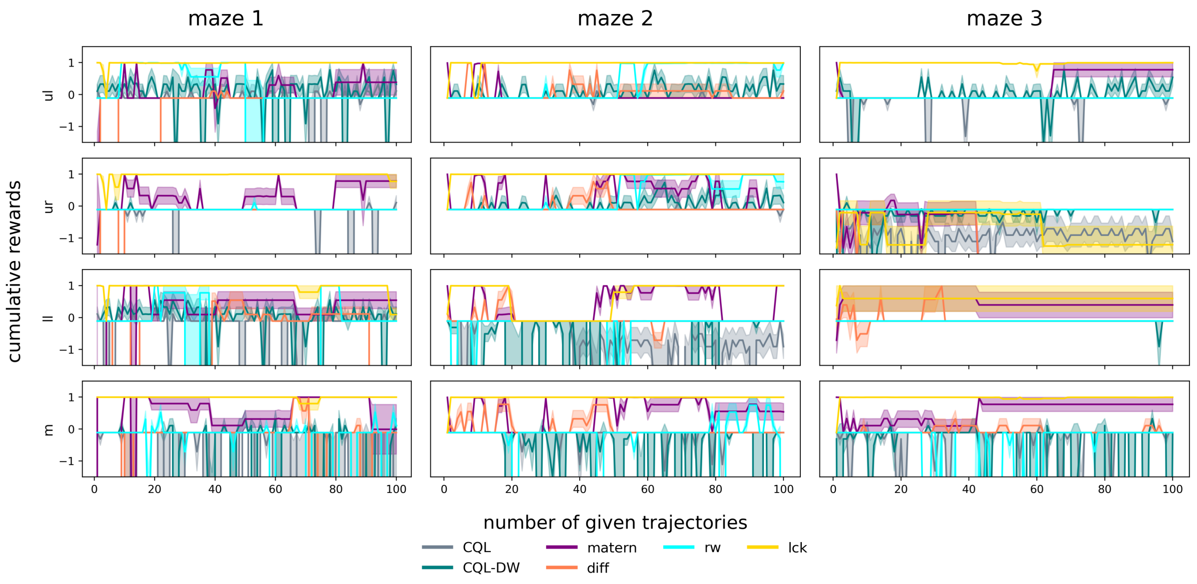

Figure 4.2: Average cumulative rewards and standard error, using suboptimal training trajectories (recorded after 400 iterations of tabular Q-learning) with step counts 56.3 ± 13.6, 96.0 ± 18.3, and 84.4 ± 18.9 for maze 1, 2, and 3, respectively.

the step limit (1000). We omit CQL as it is outperformed by CQL-DW in this particular run.

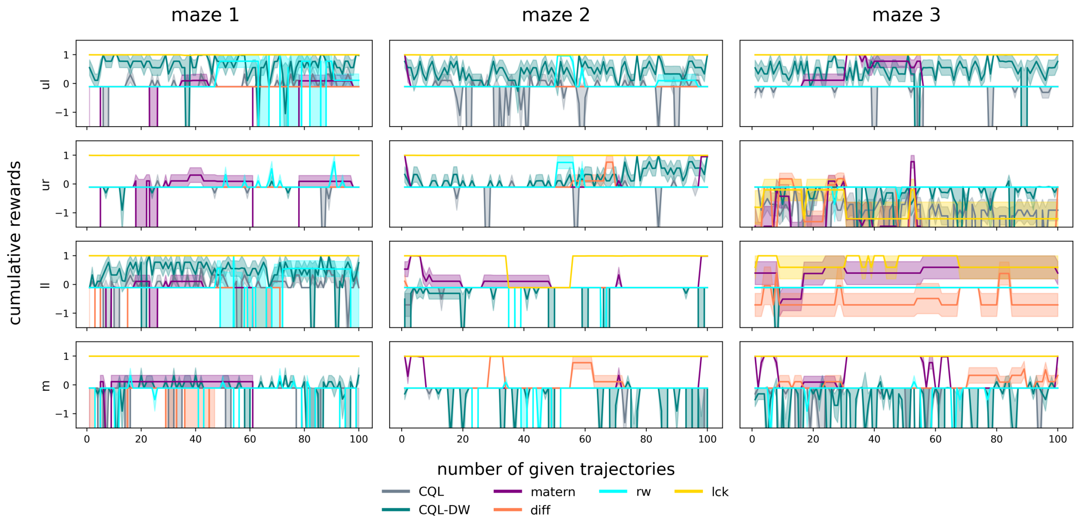

Figure 4.3: Average cumulative rewards and standard error, using suboptimal training trajectories (recorded after 700 iterations of tabular Q-learning) with step counts 39.6 ± 6.7, 68.2 ± 9.7, and 59.8 ± 10.0 for maze 1, 2, and 3, respectively.

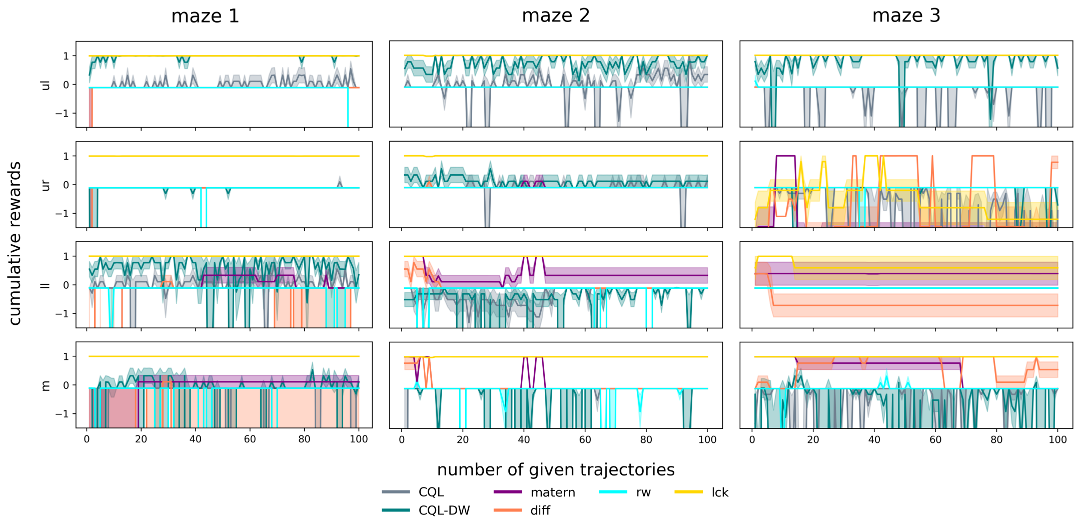

Figure 4.4: Average cumulative rewards and standard error, using suboptimal training trajectories (recorded after 1000 iterations of tabular Q-learning) with step counts 35.7 ± 6.5, 63.2 ± 9.4, and 54.2 ± 8.1 for maze 1, 2, and 3, respectively.

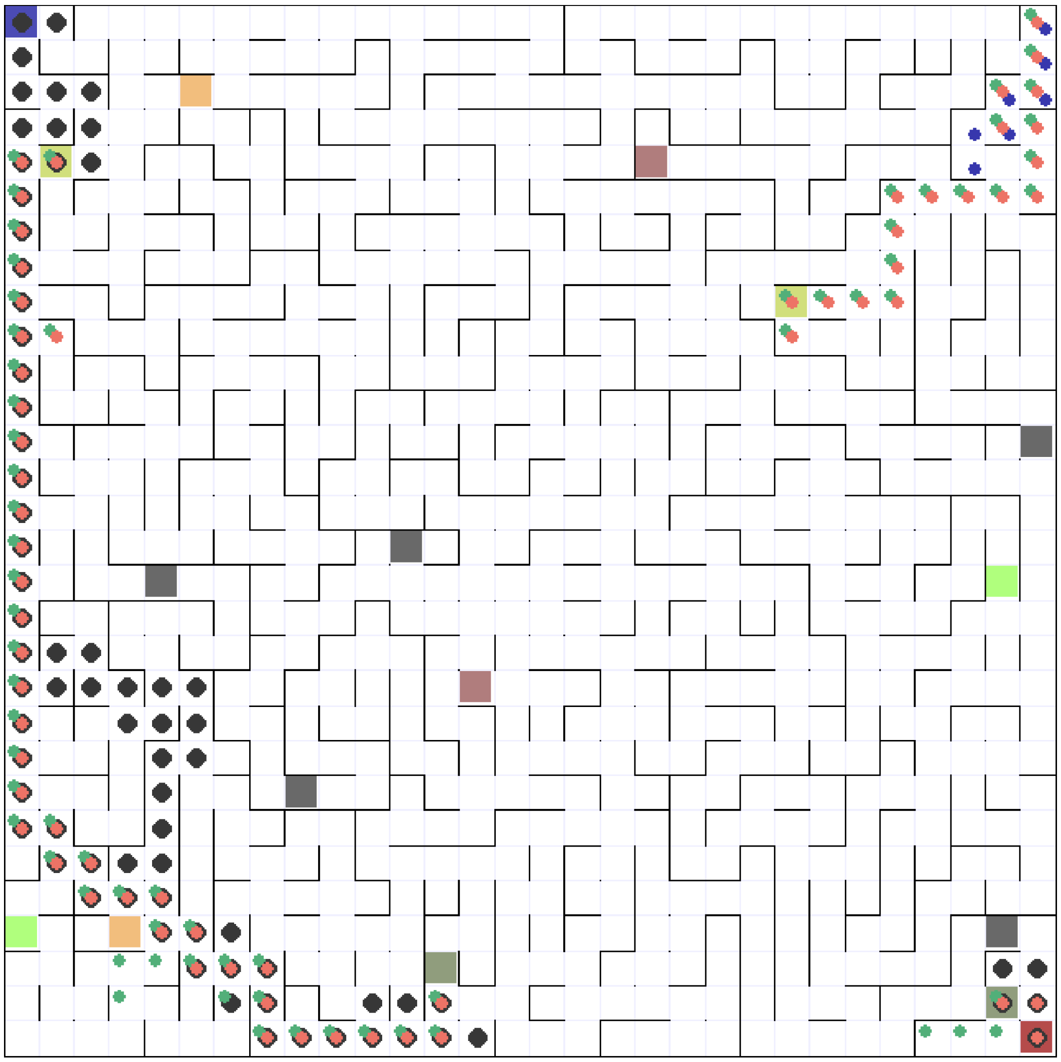

Figure 4.5: A visualization of the maze with seed 1. The black dots are cells visited by the five given trajectories, the blue dots are cells visited by CQL-DW, the green dots are cells visited by the Matérn model, and the red dots are cells visited by our model. Our model successfully reaches the goal state in 80 steps with a reward of 0.99 (for reference, the average length of the five given trajectories starting from *upper-left* is 63 steps with a variance of 14, and the average reward is 0.99), whereas both CQL-DW and the Matérn kernel failed to reach the goal state up until the step limit (1000).

Chapter 5

Conclusion

In this work, we propose a type of kernels on graphs for Gaussian processes which incorporates information about linear dependencies between target function values at different nodes. These constructed kernels, in contrast with Laplacian-based kernels, can incorporate directed information into their structure. Meanwhile, the graph Matérn kernels with certain parameters can be seen as a special case of this construction, demonstrating the generality of our construction. We also analyze theoretically and empirically the robustness of this construction under misspecified linear dependencies. To illustrate the usefulness of this construction, we showcased how to derive linear dependencies in traffic speed prediction and offline reinforcement learning settings. In both applications, these kernels significantly improve performance and data-efficiency compared to alternatives that only include prior information about connectivity. In addition, the traffic speed prediction task illustrates that this construction is especially useful for tasks on graphs where directions are essential, and the offline RL task further demonstrates robustness of our method.

The main limitations of this work and possible directions for future work are as follows.

- In the experiments, we derive task-specific linear dependencies. It would be useful to develop a general principle for inferring the linear dependencies from data.
- We assumed additive observational noise on the nodes. It would be interesting to study how our model changes with other assumptions, such as having noise on the graph structure.
- Our experiments do not make use of the uncertainty quantification inherent in Gaussian processes. It would be worth exploring our kernel in a risk-management setting, where both predictive mean and predictive variance of the GP can be utilized.

- It would be interesting to explore more diverse choices of base kernels, for example such that they take into account the uncertainty in the linear dependencies.
- When applied to RL problems, we are limited to environments with discrete states and actions. It would be interesting to explore applications to continuous spaces by discretization. Furthermore, it would be challenging but very useful to generalize our method so that it is directly applicable to continuous spaces.

References

[1] Albert-László Barabási and Márton Pósfai. *Network science.* Cambridge University Press, Cambridge, 2016.

[2] Andreas Besginow and Markus Lange-Hegermann. Constraining gaussian processes to systems of linear ordinary differential equations. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022.

[3] Viacheslav Borovitskiy, Iskander Azangulov, Alexander Terenin, Peter Mostowsky, Marc Peter Deisenroth, and N. Durrande. Matern gaussian processes on graphs. In *International Conference on Artificial Intelligence and Statistics*, 2020.

[4] Michael M. Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: Going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

[5] M. Broucke and P. Varaiya. A theory of traffic flow in automated highway systems. *Transportation Research Part C: Emerging Technologies*, 4(4):181–210, 1996.

[6] Peter Bühlmann, Jonas Peters, and Jan Ernest. CAM: Causal additive models, high-dimensional order search and penalized regression. *The Annals of Statistics*, 42(6):2526 – 2556, 2014.

[7] Bingqing Chen, Priya L. Donti, Kyri Baker, J. Zico Kolter, and Mario Bergés. Enforcing policy feasibility constraints through differentiable projection for energy optimization. In *Proceedings of the Twelfth ACM International Conference on Future Energy Systems*, e-Energy '21, page 199–210, New York, NY, USA, 2021. Association for Computing Machinery.

[8] Chao Chen, Karl Petty, Alexander Skabardonis, Pravin Varaiya, and Zhanfeng Jia. Freeway performance measurement system: Mining loop detector data. *Transportation Research Record*, 1748(1):96–102, 2001.

[9] Lu Chen, Bowen Tan, Sishan Long, and Kai Yu. Structured dialogue policy with graph neural networks. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 1257–1268, Santa Fe, New Mexico, USA, August 2018. Association for Computational Linguistics.

[10] N.A.C. Cressie. *Statistics for Spatial Data*. A Wiley-interscience publication. J. Wiley, 1991.

[11] Hanjun Dai, Yujia Li, Chenglong Wang, Rishabh Singh, Po-Sen Huang, and Pushmeet Kohli. Learning transferable graph exploration. In *NeurIPS*, pages 2514–2525, 2019.

[12] David Duvenaud. *Automatic Model Construction with Gaussian Processes*. PhD thesis, Computational and Biological Learning Laboratory, University of Cambridge, 2014.

[13] Aasa Feragen, Francois Lauze, and Soren Hauberg. Geodesic exponential kernels: When curvature and linearity conflict. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3032–3042, 06 2015.

[14] Christian Fiedler, Carsten W. Scherer, and Sebastian Trimpe. Practical and rigorous uncertainty bounds for gaussian process regression. In *AAAI Conference on Artificial Intelligence*, 2021.

[15] Allan Gut. *An Intermediate Course in Probability*. Springer Publishing Company, Incorporated, 2nd edition, 2009.

[16] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using networkx. In Gaël Varoquaux, Travis Vaught, and Jarrod Millman, editors, *Proceedings of the 7th Python in Science Conference*, pages 11 – 15, Pasadena, CA USA, 2008.

[17] Trey Ideker, Owen Ozier, Benno Schwikowski, and Andrew F. Siegel. Discovering regulatory and signalling circuits in molecular interaction networks. *Bioinformatics*, 18 Suppl 1:S233–40, 2002.

[18] Jiechuan Jiang, Chen Dun, Tiejun Huang, and Zongqing Lu. Graph convolutional reinforcement learning. In *International Conference on Learning Representations*, 2020.

[19] Carl Jidling, Niklas Wahlström, Adrian Wills, and Thomas B Schön. Linearly constrained gaussian processes. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[20] Motonobu Kanagawa, Philipp Hennig, Dino Sejdinovic, and Bharath K. Sriperumbudur. Gaussian processes and kernel methods: A review on connections and equivalences. *CoRR*, abs/1807.02582, 2018.

[21] Mohammad Kolahdouzan and Cyrus Shahabi. Voronoi-based k nearest neighbor search for spatial network databases. VLDB '04, page 840–851. VLDB Endowment, 2004.

[22] Risi Imre Kondor and John D. Lafferty. Diffusion kernels on graphs and other discrete input spaces. In *Proceedings of the Nineteenth International Conference on Machine Learning*, ICML '02, page 315–322, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc.

[23] Nils M. Kriege, Fredrik D. Johansson, and Christopher Morris. A survey on graph kernels. *Applied Network Science*, 5:1–42, 2019.

[24] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS'20, Red Hook, NY, USA, 2020. Curran Associates Inc.

[25] Markus Lange-Hegermann. Linearly constrained gaussian processes with boundary conditions. In Arindam Banerjee and Kenji Fukumizu, editors, *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, volume 130 of *Proceedings of Machine Learning Research*, pages 1090–1098. PMLR, 13–15 Apr 2021.

[26] Miguel Lázaro-Gredilla and Aníbal Figueiras-Vidal. Inter-domain gaussian processes for sparse inference using inducing features. In Y. Bengio, D. Schuurmans, J. Lafferty, C. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems*, volume 22. Curran Associates, Inc., 2009.

[27] David Lazer, Alex Pentland, L. Adamic, Sinan Aral, Albert-Laszlo Barabasi, Devon Brewer, Nicholas Christakis, Noshir Contractor, Jessica Fowler, and Myron Gutmann. Life in the network: The coming age of computational social science. 323, 01 2009.

[28] Haitao Liu, Yew-Soon Ong, Xiaobo Shen, and Jianfei Cai. When gaussian process meets big data: A review of scalable gps. *IEEE Transactions on Neural Networks and Learning Systems*, 31(11):4405–4423, 2020.

[29] Danielle C. Maddix, Nadim Saad, and Yuyang Wang. Modeling advection on directed graphs using matérn gaussian processes for traffic flow. *CoRR*, abs/2201.00001, 2022.

[30] Sephora Madjiheurem and Laura Toni. Representation learning on graphs: A reinforcement learning application. In *Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 89 of *PMLR*, 2019.

[31] Sephora Madjiheurem and Laura Toni. Representation learning on graphs: A reinforcement learning application. In Kamalika Chaudhuri and Masashi Sugiyama, editors, *Proceedings of the Twenty-Second International Conference on Artificial Intelligence and Statistics*, volume 89 of *Proceedings of Machine Learning Research*, pages 3391–3399. PMLR, 16–18 Apr 2019.

[32] Alexander G. de G. Matthews, Mark van der Wilk, Tom Nickson, Keisuke. Fujii, Alexis Boukouvalas, Pablo León-Villagrá, Zoubin Ghahramani, and James Hensman. GPflow: A Gaussian process library using TensorFlow. *Journal of Machine Learning Research*, 18(40):1–6, apr 2017.

[33] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. In *NIPS Deep Learning Workshop*. 2013.

[34] Yin Cheng Ng, Nicolò Colombo, and Ricardo Silva. Bayesian semi-supervised learning with graph gaussian processes. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 1690–1701, Red Hook, NY, USA, 2018. Curran Associates Inc.

[35] OpenStreetMap contributors. Planet dump retrieved from https://planet.osm.org . `https://www.openstreetmap.org`, 2017.

[36] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, page 701–710, New York, NY, USA, 2014. Association for Computing Machinery.

[37] Marvin Pförtner, Ingo Steinwart, Philipp Hennig, and Jonathan Wenger. Physics-informed Gaussian process regression generalizes linear PDE solvers, 2022.

[38] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005.

[39] Havard Rue and Leonhard Held. *Gaussian Markov Random Fields: Theory And Applications (Monographs on Statistics and Applied Probability)*. CRC Press, 2005.

[40] Alexander J Smola and Risi Kondor. Kernels and regularization on graphs. In *Learning Theory and Kernel Machines: 16th Annual Conference on Learning Theory and 7th Kernel Workshop, COLT/Kernel 2003, Washington, DC, USA, August 24-27, 2003. Proceedings*, pages 144–158. Springer, 2003.

[41] O. Sorkine, D. Cohen-Or, Y. Lipman, M. Alexa, C. Rössl, and H.-P. Seidel. Laplacian surface editing. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, SGP '04, page 175–184, New York, NY, USA, 2004. Association for Computing Machinery.

[42] G. W. Stewart. On the perturbation of pseudo-inverses, projections and linear least squares problems. *SIAM Review*, 19(4):634–662, 1977.

[43] Gilbert Strang. *Introduction to Linear Algebra.* Wellesley-Cambridge Press, Wellesley, MA, fourth edition, 2009.

[44] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction.* MIT press, 2018.

[45] Laura P. Swiler, Mamikon Gulian, Ari L. Frankel, Cosmin Safta, and John D. Jakeman. A survey of constrained gaussian process regression: Approaches and implementation challenges. *Journal of Machine Learning for Modeling and Computing*, 1(2):119–156, 2020.

[46] Josephine Thomas, Alice Moallemy-Oureh, Silvia Beddar-Wiesing, and Clara Holzhüter. Graph neural networks designed for different graph types: A survey. *Transactions on Machine Learning Research*, 2023.

[47] Felipe Tobar, Thang D Bui, and Richard E Turner. Learning stationary time series using gaussian processes with nonparametric kernels. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[48] Csaba Toth and Harald Oberhauser. Bayesian learning from sequential data using Gaussian processes with signature covariances. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 9548–9560. PMLR, 13–18 Jul 2020.

[49] Sam Toyer, Felipe W. Trevizan, Sylvie Thiébaux, and Lexing Xie. Action schema networks: Generalised policies with deep learning. In *AAAI*, 2018.

[50] Arun Venkitaraman, Saikat Chatterjee, and Peter Handel. Gaussian processes over graphs. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5640–5644, 2020.

[51] Daniel Kudenko Vikram Waradpande and Megha Khosla. Graph-based state representation for deep reinforcement learning. In *Proceedings of the 16th International Workshop on Mining and Learning with Graphs (MLG)*, 2020.

[52] S.V.N. Vishwanathan, Nicol N. Schraudolph, Risi Kondor, and Karsten M. Borgwardt. Graph kernels. *Journal of Machine Learning Research*, 11(40):1201–1242, 2010.

[53] Tingwu Wang, Renjie Liao, Jimmy Ba, and Sanja Fidler. Nervenet: Learning structured policy with graph neural networks. In *International Conference on Learning Representations*, 2018.

[54] Xiao Wang, Hongrui Liu, Chuan Shi, and Cheng Yang. Be confident! towards trustworthy graph neural networks via confidence calibration. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021.

[55] Max A Woodbury. Inverting modified matrices. In *Memorandum Rept. 42, Statistical Research Group*, page 4. Princeton Univ., 1950.

[56] H. Yuan, H. Yu, S. Gui, and S. Ji. Explainability in graph neural networks: A taxonomic survey. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 45(05):5782–5799, may 2023.

[57] Yin-Cong Zhi, Yin Cheng Ng, and Xiaowen Dong. Gaussian processes on graphs via spectral kernel learning. *IEEE Transactions on Signal and Information Processing over Networks*, 9:304–314, 2023.

APPENDICES

Appendix A

Matrix identities

Woodbury matrix identity For matrices $\mathbf{A} \in \mathbb{R}^{n\times n}$, $\mathbf{U} \in \mathbb{R}^{n\times k}$, $\mathbf{C} \in \mathbb{R}^{k\times k}$, and $\mathbf{V} \in \mathbb{R}^{k\times n}$, the *Woodbury matrix identity* [55] states that

$$(\mathbf{A} + \mathbf{UCV})^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{U}(\mathbf{C}^{-1} + \mathbf{VA}^{-1}\mathbf{U})^{-1}\mathbf{VA}^{-1} \tag{A.1}$$

when all relevant inverses exist.

Matrix square root Let $\mathbf{A} \in \mathbb{R}^{n\times n}$ be a positive-semidefinite matrix. By definition, $\mathbf{A}$ is symmetric and thus has an eigendecomposition according to the spectral theorem [43]:

$$\mathbf{A} = \sum_{i=1}^{n} \lambda_i \boldsymbol{v}_i \boldsymbol{v}_i^T \quad \text{where } \lambda_i \in \mathbb{R} \text{ and } \lambda_i \geq 0.$$

Then, the square root of $\mathbf{A}$ is defined as

$$\mathbf{A}^{1/2} := \sum_{i=1}^{n} \lambda_i^{1/2} \boldsymbol{v}_i \boldsymbol{v}_i^T. \tag{A.2}$$

One can verify that $\mathbf{A}^{1/2}$ is symmetric and $\mathbf{A}^{1/2}\mathbf{A}^{1/2} = \mathbf{A}$.

Appendix B

Experimental details

Our GP models are implemented with GPflow [32]. The baseline Matérn kernel has three trainable parameters, with initial values $\nu = 3/2$, $\kappa = 5$, and $\sigma_f = 1$. The baseline diffusion kernel has two trainable parameters, with initial values $\kappa = 5$ and $\sigma_f = 1$. The baseline random walk kernel has parameters $\alpha = 0.5$ and $p = 10$ determined by grid search, and trainable parameter σ_f set to be 1 initially. Our kernel has trainable parameter σ_f, initially set to be 1. Below we provide experiment-specific details.

B.1 Synthetic Data

The random graphs we experiment on are (1) a Barabási–Albert graph with 500 nodes and 3 edges per node, (2) an Erdős–Rényi graph with 500 nodes and parameter $p = 0.2$, and (3) a Gaussian random partition graph with 500 nodes, mean cluster size 5, shape parameter 5, in-cluster probability 0.2, and out-of-cluster probability 0.1. All three graphs are generated using NetworkX [16] graph generators with random seed 1. Experiment results on graphs generated with other seeds are very similar.

During training, we conduct ten runs with seeds $[1, 2, \ldots, 10]$, where the randomness affects how training nodes are chosen. Each model is trained for 1000 iterations.

B.2 Traffic Speed Prediction

We use the same experiment setup as in [3]. More specifically, we use the same snapshot speed data from PeMS [8] as this work. As a result, we do not utilize the temporal aspects of the data, and we do not compare against GNNs for temporal traffic forecasting.

For the four models that make use of the linear dependencies in Equation (14), we do a rough grid search among $[0.85, 0.90, 0.95, 0.99999]$ to determine a good value for the scalar b in our linear dependencies. It turns out 0.95 is the best value for all models.

We conduct ten runs with seeds $[1, 2, \dots, 10]$, where the randomness affects how training nodes are chosen. Each model is trained for 12000 iterations.

B.3 Offline Reinforcement Learning

B.3.1 Environment Details

The three 30×30 mazes are generated with random seeds $[1, 2, 3]$. In the environment, the states are the cells in the maze, and the actions are going up, down, left, and right. When an agent performs an action in a certain direction, there is a 0.1 probability to transit to each lateral direction. For example, if an agent executes an action to go down, with 0.8 probability it will transit to the cell below, with 0.1 probability each it will transit to the cells on the left or right. When an agent tries to move in a direction where there is a wall, it stays at the current cell. There are five pairs of portals in the maze. If an agent reaches a portal in a pair, it will be directly teleported to the other portal in the pair. The goal state is the cell in the lower right corner, with a reward of 1.0. There are five traps in the maze, each with a reward of -1.0. All other states have a reward of $-0.1/900$. In order to maximize the cumulative rewards, an agent would try to reach the goal state using as few steps as possible while avoiding traps. Note that there are potentially many paths to reach the goal state.

B.3.2 Offline Trajectories collection

We collect sub-optimal trajectories going from the upper left corner to the goal state, by recording trajectories of a tabular Q-learning agent for 100 consecutive episodes after 700, 1000, or 3000 episodes of training. Statistics of the trajectories are given in the main text.

B.3.3 Training and Evaluation Details

We use a linearly constrained kernel with base kernel $0.1\mathbf{I}$. Our baseline CQL model has a hidden layer of size 256. The input dimension is 2 for CQL, and 30 for CQL-DW with embedding dimension 30. We train CQL-based models for 50 episodes with batch size 128, and GP models for 2000 iterations. During evaluation, for each datapoint, we conduct 5 runs with random seeds $[1, 2, \ldots, 5]$, where randomness affects stochasticity of the maze.

We empirically compare the training time for the four models using line_profiler. The training time for CQL, CQL-DW, and the GP model with Matérn kernel are similar. The training time for the GP model with our kernel is roughly half of those for other models. Generating node embeddings for CQL-DW takes additional time that is of the same order as the training time, and so does converting the value estimates of the GP into Q-values. A more vectorized implementation of our method might have much shorter conversion time.