1 Publication-style review

1.1 Summary

The paper [Bor+21] introduces a new type of kernel for a Gaussian process on a weighted graph. In particular, the popular Matérn kernel for Gaussian processes on Euclidean space is in some sense converted to a graph-theoretic kernel by replacing the Euclidean Laplace operator with the graph-theoretic Laplacian matrix. Mathematical details of the derivation are not provided in significant detail. As computational efficiency when working with large datasets is a general concern when using Gaussian processes, the authors also discuss computational aspects of the graph Matérn Gaussian process. Sparse graphs generally lead to faster computation, and various approximation techniques are also discussed, both for regression and classification. The authors also provide two comparisons of graph Matérn kernel with other kernels, and the performance is roughly comparable. It is not clear when one kernel might be preferable to another.

1.2 High-level comments on strengths and weaknesses

The main strength of the paper is the introduction of a new (Matérn) kernel for Gaussian processes on graphs. Furthermore, it is at least of theoretical interest that the new kernel is so closely related to the popular Matérn kernel for Gaussian processes on Euclidean space. The main weaknesses of the paper are twofold. First, the graph Matérn kernel does not appear to meaningfully outperform pre-existing alternatives, although this is not a significant weakness. Second, and perhaps slightly more signifcant, the the mathematical detail in [Bor+21] is relatively scarce and it is not obvious that the target audience will be able to reconstruct the mathematical derivations. In relation to other work, [Bor+21] is quite similar at least in inspiration to the authors' previous work [Bor+20], which applies a very similar trick in the replacement of the Euclidean Laplace operator with a related Laplace operator—the Laplace-Beltrami operator in the case of [Bor+20], and the graph Laplacian in the case of [Bor+21].

1.3 Originality

It is not completely obvious precisely what is novel in the paper in question, because there doesn't appear to be any phrase along the lines of "We present a novel. . . " or "We introduce. . . " that would make clear exactly what novel work has been done by the authors. It *appears* that, while Gaussian processes on Euclidean space are well known and Gaussian processes on graphs have been studied, the paper in question is the first to essentially port the Matérn kernel version of a Gaussian process from Eucliean space to the graph context. The work is a combination of prior techniques in the sense that both Gaussian processes on graphs, and Matérn kernels on Euclidean space, have both been studied (separately, apparently). If the work in question is the first to introduce Matérn Gaussian processes on graphs, then it obviously differentiates itself from prior work. Regarding the question of adequate citations, I think it depends to a significant extent on the novelty of the work. If the paper *is* the first to introduce Matérn Gaussian processes on graphs, then by dint of its originality there will not be as as much prior work to cite. If it is not, then the appears to suffer from inadequate citations reflecting this fact, particularly in Sections 2.2 and 3.

1.4 Quality

It is important to keep one's audience in mind when presenting new work. The lead author is a pure mathematician by training and is presenting his work to an audience presumably consisting of statisticians, so there is presumably a large gap in mathematical knowledge beween the author and his audience. It seems very unlikely that many statisticians will have any particular familiarity with, for example, stochastic partial differential equations, heat semigroups, or Riemannian manifolds. Given this gap in mathematical knowledge it appears that the paper in question may be short in detail for some of its mathematical derivations, for example precisely how the standard Matérn kernel for $\mathbb{R}^n$ is converted to to a graph-theoretic version. If statisticians are happy to ignore technical details then the paper might be considered technically sound, especially if we presume that the lead author's mathematical training reduces the chance of his results being erroneous. On the other hand, if the audience is interested in understanding precisely how the graph-theoretic Matérn kernel was derived, then the paper appears short on mathematical justifications. Indeed, the word "proof" does not seem to appear in the paper. The authors appear to honestly present the strengths and weaknesses of their work insofar as they do not claim that Matérn Gaussian processes necessarily represent a major advance, especially since their own empirical results do not appear to show significantly improved performance compared to other kernels.

1.5 Clarity

The writing and presentation generally seem clear, modulo the aforementioned potential issues with a lack of mathematical clarity. The question of whether the paper "adequately informs the reader" of course depends on the reader's level of mathematical sophistication—see the above discussion on this point. Regarding code, the authors have provided code in a GitHub repository.

1.6 Significance

The results appear to be important insofar as they present a new kernel for Gaussian processes on graphs, and in particular it is interesting that a common kernel for Euclidean space has been "ported" over to graphs. The mathematical techniques used appear closely related (at least in spirit) to the techniques employed in the authors' previous paper [Bor+20]

It is unclear whether the mathematical techniques employed in the paper are ripe for further exploitation, at least in part because I don't fully understand then, nor are they thoroughly described. The authors' own experiments do not suggest that graph Matérn Gaussian processes blow away the competition, but with only two comparisons provided, it is too early to judge. At any rate, requiring that a new technique supercede all existing techniques is too high a bar. It is not clear that the authors provide "unique data" or "unique conclusions about existing data", and as described above, it is hard to evaluate their theoretical approach.

1.7 List of questions for the authors

1. Where and how can I learn more about the derivations and convergence results in this paper?

2. Do you have any expectations regarding when the Matérn graph Gaussian process might perform better or worse than alternative kernels, both in terms of accuracy and computational efficiency? (Might the answer to this question depend both on the characteristics of the graph, the dataset itself and the desired type of prediction?)

3. What if anything is impeding the application of Matérn kernels for Gaussian processes on directed and/or infinite graphs?

1.8 Technical/methodological summary

1.8.1 Related literature and broader context

A stochastic process X on parameter space T is said to be *Gaussian* if the random variable $\sum_{i=1}^{n} c_i X_{t_i}$ is Gaussian, for any choice of $n \in \mathbb{N}$, $t_1, \ldots, t_n \in T$, and $c_1, \ldots, c_n \in \mathbb{R}$ [Kal21]. Gaussian processes have been studied theoretically for at least a century, since the archetypal stochastic process, Brownian motion, is itself Gaussian [Kal21]. However, it appears that only in the last few decades have Gaussian processes become a significant tool in statistics and machine learning [RW06].

By Lemma 14.1 of [Kal21], the distribution of a Gaussian process X is uniquely determined by the functions

$$m_t = \mathrm{E}X_t, \quad r_{s,t} = \mathrm{Cov}(X_s, X_t) \tag{1}$$

for $s, t \in T$. However, among applied practitioners it appears that when using Gaussian processes, much more attention is paid to the covariance function than the mean function [Gar23]. (The precise details regarding the relationship between the regular covariance operation as in (1), and covariance functions more generally, will not be explored in this report.)

The standard definition of a stochastic process is a sequence of random variables $X = \{X_t : t \in T\}$ over a parameter space T, which only requires the existence of some original probability space $(\Omega, \mathcal{F}, \mathbb{P})$ and a (measurable) state space $(S, \mathcal{S})$. This abstract definition permits the possibility that a stochastic process may be defined not only on Euclidean space, but also on other spaces, such as graphs [Gri18]. Gaussian processes in particular are typically employed with Euclidean space as domain and codomain, but there is nothing in principle preventing Gaussian processes being defined on the vertices of a graph $G = (V, E)$, i.e. allowing $\Omega = V$. Indeed, this has been explored, as for example in [KL02]. One of the key challenges in using Gaussian processes on graphs has been finding proper covariance functions, as explained in [KL02], since covariance functions (or "kernels") are required to be symmetric and positive semi-definite.

In the work [Bor+21], a new kernel for Gaussian processes on graphs is explored, in some sense deriving a graph-theoretic version of the popular Matérn kernel for Gaussian processes on Euclidean space. The Euclidean Matérn kernel is a function $C_\nu : \mathbb{R}^d \to \mathbb{R}$ given by

$$C_\nu(d) = \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\sqrt{2\nu} \frac{d}{\ell} \right)^{\nu} K_\nu \left(\sqrt{2\nu} \frac{d}{\ell}, \right), \tag{2}$$

where d is the Euclidean distance between points $\boldsymbol{x}, \boldsymbol{x}' \in \mathbb{R}^n$; $\nu, \ell > 0$; Γ refers to the gamma function and K_ν is the modified Bessel function of the second kind [RW06]. In 1.8.2, we will explore the derivation of the Matérn kernel for Gaussian processes on graphs.

(The paper [Bor+21] also discusses a graph-theoretic version of the squared exponential kernel, which is related to the Matérn kernel. However, due to space limitations, this report will focus on the Matérn kernel.)

1.8.2 Proposed methodology and theoretical properties

There does not appears to be a "proposed methodology" in [Bor+21], insofar as the main contribution is a new kernel for Gaussian processes on graphs—presumably any existing method of using Gaussian processes on graphs can be used with the new kernel. Therefore, this section will focus on explaining whatever theoretical aspects of [Bor+21] that can reasonably be explained, given space limitations.

The paper [Bor+21] refers to the paper [Whi63] in support of a claim that a Gaussian process f on $\mathbb{R}^d$ with Matérn kernel as in (2) satisfies the stochastic partial differential equation

$$\left(\frac{2\nu}{\kappa^2} - \Delta \right)^{\frac{\nu}{2} + \frac{d}{4}} f = \mathcal{W}, \tag{3}$$

where ν and κ are parameters, Δ is the Laplace operator and $\mathcal{W}$ is Gaussian white noise. Unfortunately, the paper [Whi63] cannot be located in either the Simon Fraser University Library, or the University of British Columbia library, so this claim cannot investigated in further detail.

In order to derive a graph-theoretic version of the Matérn kernel, the authors of [Bor+21] focus on the characterization of a Gaussian process as in (3). Although the derivation is not totally clear, a key step in the derivation appears to be replacing the Euclidean Laplace operator Δ with the Laplacian matrix $\boldsymbol{\Delta}$, which is a graph-theoretic analogue of the Euclidean Laplace operator Δ. (Indeed, this appears to be a very similar at least in spirit to the maneuver used by the authors in their previous work [Bor+20], in which they replace the Euclidean Laplace operator in (3) with its Riemannian generalization, the Laplace-Beltrami operator. This substitution leads to the development of a Matérn Gaussian process on Riemannian manifolds in [Bor+20].)

The derivation can be at least partly explained as follows. First assume that the function $\Phi\colon \mathbb{R} \to \mathbb{R}$ as defined in [Bor+21] is analytic, roughly meaning that it is equal to its Taylor series, so we can write

$$\Phi(z) = a_0 + a_1 z + a_2 z^2 + \dots . \tag{4}$$

If we then consider a diagonal matrix

$$\mathbf{D} = \begin{pmatrix} d_1 & 0 & 0 & \dots & 0 \\ 0 & d_2 & 0 & \dots & 0 \\ 0 & 0 & d_3 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & d_n \end{pmatrix},$$

and we plug this into Φ, then we can write

$$\begin{aligned} \Phi(\mathbf{D}) &= a_0 + a_1\mathbf{D} + a_2\mathbf{D}^2 + \dots \\ &= a_0\mathbf{I} + a_1 \begin{pmatrix} d_1 & 0 & 0 & \dots & 0 \\ 0 & d_2 & 0 & \dots & 0 \\ 0 & 0 & d_3 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & d_n \end{pmatrix} + a_2 \begin{pmatrix} d_1^2 & 0 & 0 & \dots & 0 \\ 0 & d_2^2 & 0 & \dots & 0 \\ 0 & 0 & d_3^2 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & d_n^2 \end{pmatrix} + \dots \\ &= \begin{pmatrix} \Phi(d_1) & 0 & 0 & \dots & 0 \\ 0 & \Phi(d_2) & 0 & \dots & 0 \\ 0 & 0 & \Phi(d_3) & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & \Phi(d_n) \end{pmatrix}. \end{aligned} \tag{5}$$

Now, if $\mathbf{O}$ is an orthogonal matrix, then note that

$$\begin{aligned} (\mathbf{ODO}^\intercal)^n &= \underbrace{(\mathbf{ODO}^\intercal)\dots(\mathbf{ODO}^\intercal)}_{n \text{ times}} \\ &= \mathbf{OD}^n\mathbf{O}^\intercal, \end{aligned}$$

so we conclude that

$$\begin{aligned} \Phi(\boldsymbol{\Delta}) &= \Phi(\mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^\intercal) \\ &= a_0 + a_1\mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^\intercal + a_2(\mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^\intercal)^2 + \dots \\ &= \mathbf{U}\Phi(\boldsymbol{\Lambda})\mathbf{U}^\intercal, \end{aligned}$$

which is consistent with (9) in [Bor+21].

The remainder of the argument, namely proceeding from (10) to (11) in [Bor+21], is not totally clear, since it's not obvious where the matrices $\mathbf{U}$ and $\mathbf{U}^\intercal$ have gone. However, it seems possible that their disappearance is related to the invariance of $\boldsymbol{\mathcal{W}} \sim \mathrm{N}(\mathbf{0}, \mathbf{I})$ under orthogonal transformations.

Based on (12) in [Bor+21], it appears that the final form of the graph Matérn kernel is

$$\left(\frac{2\nu}{\kappa^2} + \mathbf{\Delta}\right)^{-\nu}. \tag{6}$$

I believe this ought to be interpreted as follows. First note that (10) in [Bor+21], we considered the function $\Phi \colon \mathbb{R} \to \mathbb{R}$ given by

$$\Phi(\lambda) = \left(\frac{2\nu}{\kappa^2} + \lambda\right)^{\frac{\nu}{2}}. \tag{7}$$

Note that (6) can be read as $\Phi(\mathbf{\Delta})$, and in (9) in [Bor+21], we were instructed to interpret $\Phi(\mathbf{\Delta})$ as $\mathbf{U}\Phi(\mathbf{\Lambda})\mathbf{U}^\intercal$, where $\mathbf{\Delta} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\intercal$. Therefore, (6) seems to represent at least in part the addition of $2\nu/\kappa^2$ to each eigenvalue of $\mathbf{\Delta}$. (It is unclear how the exponent changed from $\nu/2$ to $-\nu$ between (7) and (6).)

1.8.3 Computational complexity, crucial aspects and potential bottlenecks

Gaussian processes, in and of themselves, are simply stochastic processes as described earlier, so it may only make sense to discuss computational performance in the context of a specific manner of use of Gaussian processes. As such, since [Bor+21] is more focused on introducing a new flavor of Gaussian process, i.e. one with a Matérn kernel on a weighted graph, as opposed to prescribing a specific manner of use, we may begin with generalities and then discuss specific details described in [Bor+21].

Use of Gaussian processes often involves matrix inversion [BDT13; Liu+20], which is a key computational bottleneck in their use. Inversion of an $n \times n$ matrix is generally $O(n^3)$, although faster algorithms, such as Strassen's algorithm, exist [Cor+22]. It is presumably difficult to draw an objective line beyond which datasets become too big for rote use of Gaussian processes, but computational limitations are clearly significant enough that references on Gaussian processes and their use devote attention to efficiency and approximation techniques [RW06; Gar23].

Regarding the specific discussion of computational issues in [Bor+21], a few issues are mentioned, and they are presented below.

First, when the graph G is sparse (having relatively few edges), the graph Laplacian $\mathbf{\Delta}$ is a sparse matrix (having relatively few non-zero entries). This leads to sparse precision matrices, which are inverses of covariance matrices [Gar23], and special techniques have been developed to take advantage of matrix sparsity [GL13].

Second, the kernel matrix can be estimated via "truncated eigenvalue expansion". The explanation of this process in [Bor+21] appears to assume familiarity with these techniques, but the essential idea appears to be as follows. First, we obtain the ℓ smallest eigenvalues and eigenvectors of the matrix $\mathbf{\Delta}$ using a method such as the Lanczos algorithm. (Apparently, the small eigenvalues of the graph Laplacian give important information about the "connectivity" of a graph [Coo16]; it is unclear whether this is relevant for the methods discussed in [Bor+21].) If we interpret the graph Matérn kernel in (6) in the $\Phi(\mathbf{\Lambda})$ sense, then the matrix (5) makes relatively clear that we will have obtained the ℓ *largest* eigenvalues of the Matérn kernel, since Φ as in (6) is a decreasing function.

According to [Bor+21], the main drawback of the truncated eigenvalue expansion approach is so-called "variance starvation", which appears to mean that approximation of a Gaussian process can lead to badly underestimated variance as the number of observations increases; see for example Figure 1 in [Wan+18].

Another method concerns modification of the parameter ν in the equation (3), which according to [Bor+21], which roughly speaking controls the smoothness of the sample paths of the Gaussian process. According to [Gar23], the parameter ν is typically given values $k + 1/2$ where $k \in \mathbb{N}$, and higher values lead to smoother sample paths. In contrast, in [Bor+21] it is suggested to set ν to a small whole number value.

The succeeding explanation is very brief, but apparently this results in a Gaussian Markov random field, for which training with large amounts of data is relatively straightforward.

For classification, use of Gaussian processes requires some modification because it's less reasonable to assume that likelihoods are Gaussian [RW06]. Again the explanation in [Bor+21] is quite sparse, but the idea appears to be to try to select a representative subset of the data, and calculate an approximation that is as close as possible to that using the entire dataset by minimizing the Kullback-Leibler divergence between two distributions. If the approximation is chosen to be Gaussian, then this amounts to choosing the closest Gaussian process posterior to the true non-Gaussian posterior. (It is not clear what sort of penalty might be paid for this distributional approximation.) According to [Bor+21], there are scalable algorithms for this approach.

1.9 Implementation

The authors of [Bor+21] provide an implementation of their methods on GitHub (link). The code appers to be intended to integrate with GPflow, a package for building Gaussian process models in Python, using TensorFlow.

1.9.1 Alternative methods that could be applied

In order to discuss "alternative methods that could also be applied to the given problem", we must attempt to define "the problem". Presumably the goal is to do some kind of machine learning on a graph. Below we discuss two cases—Gaussian processes on graphs, and other machine learning techniques on graphs.

Gaussian processes on graphs As [Bor+21] makes clear, Gaussian processes on graphs have been investigated since at least 2002 [KL02], so there are kernels available besides the graph Matérn kernel introduced in [Bor+21]. Based on the comparison provided in [Bor+21], it appears that the graph Matérn kernel is roughly comparable to earlier kernels such as the diffusion kernel, the random walk kernel, and the inverse cosine kernel. However, the comparison in [Bor+21] is not exhaustive, so perhaps a clearer picture of the advantages and disadvantages of different kernels could emerge through further investigation.

Other machine learning techniques on graphs The main methods for machine learning on graphs appear to be graph neural networks, and graph transformers [Fou23; Les23]. A brief persual of Google does not appear to return any comparisons of graph neural networks or graph transformers with Gaussian processes on graphs.

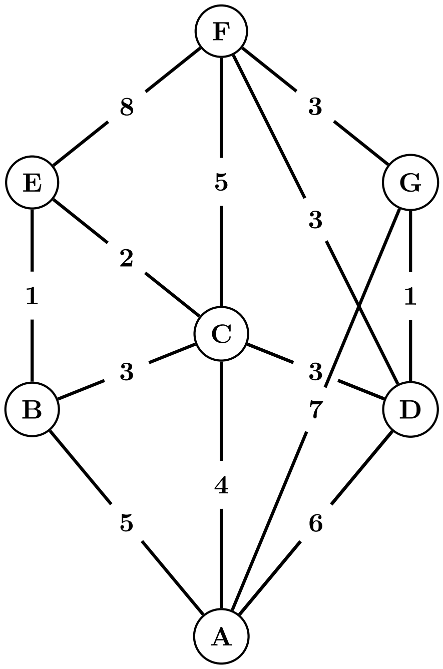

Figure 1: A weighted graph G, representing a simple example of a reinforcement learning state space $\mathcal{S}$ for the mini-project proposal in Section 2. Unlike a standard weighted graph, which only has weights on edges, we will consider a graph that also has a real number associated with each vertex. The informal idea is that a true, complete state $s \in \mathcal{S}$ will consist of all weight and vertex values and locations in the graph. In contrast, the observation $o \in \mathcal{O}$, $o \subseteq s$ will not contain the vertex weights. The exact details of the proposal are not totally clear, but the idea is that weights on the graph edges will represent costs incurred, and the vertex weights will represent rewards.

2 Mini-Proposal

The proposed project concerns an application of Gaussian processes on graphs to a Markov decision process.

2.1 Area of Opportunity

A brief Google search for “reinforcement learning graph gaussian process” did not return any results that looked comparable.

2.2 Proposed method/approach

The informal idea is that the reinforcement learning agent is managing logistics or route-planning for a concern that must move goods from an origin to a destination on a structure that can be represented by a weighted graph $G = (V, E)$. A weight w between vertices v and v' in V represents the cost of traveling from v to v', and the agent's task is to find the route from origin to destination that minimizes the cost accumulated along the way.

However, at every time step, the costs of traveling between nodes are subject to randomness, and not all costs are observable to the agent. For simplicity, we assume that the agent requires one time step to travel along an edge between nodes. This setup will require the agent to re-calculate its estimates of the costs of routes for the remaining portion of the journey at every time step. This seems to be a reasonable assumption, since it is easy to imagine a delivery van driver having to change alter his planned route because of an (unforeseen, obviously) traffic accident that causes a traffic jam along his planned route.

Even though the Markov decision process is assumed to be only partially observable, suppose the agent has access to a good estimator of its true state. This estimator could take the form of a function GP: $\mathcal{O} \to \mathcal{S}$ that, given an observation o, produces the estimate $\mathrm{GP}(o) = \hat{s}$ of s. Given the estimate $\hat{s}$ of s, the agent may be able to proceed as though its environment is in fact fully observable, and therefore use standard reinforcement learning algorithms that assume full observability. Furthermore, it may be possible train the estimator GP in an online fashion as the agent interacts with its environment, since when the agent acts on information from GP, it may receive one of the true costs it was attempting to estiamte.

As a comparison, the method of estimating the agent's true state and proceeding as though the state is known might be compared with standard techniques for partially observable Markov decision processes.

See Figure 1 for an simple example.

2.2.1 Computational implementation

Programming language I plan to use the Python language, since it is standard in machine learning.

Data structures Although the textbook [Cor+22, Ch. 20] suggests storing a weighted graph as an adjacency list, it seems more straightfoward to use matrices. The asymptotic efficiency of various representations do not seem relevant at this stage.

2.3 Expected technical challenges and bottlenecks

2.3.1 Constructing an appropriate example

One important aspect of this mini-project is finding an appropriate example for testing. An example that is too elaborate may hinder progress, while an example that is too simple may be insufficiently interesting for analysis. Some aspects of example selection requiring judgment are discussed below.

Graph design It is easy to imagine, in general, that a large business may own multiple warehouses across the country, so that in a very large graph, deliveries can in principle be made from any origin (warehouse) node to any other node in the graph. This seems excessively complicated for this mini-project, so instead it seems more reasonable to consider only a simplified graph with one origin node and one destination node, as in Figure 1.

(One simplifying assumption is that an edge between vertices v and v' can be traversed both ways, and furthermore, the costs to travel each way are not separate. This clearly an approximation to reality, since highways can be congested in one direction but not in the other direction. Fixing this assumption would require replacing every edge in a graph as in Figure 1 with two opposite directed edges, with their own cost distributions. However, this is probably an unnecessary complication for this mini-project, especially since the graph Matérn Gaussian process as in [Bor+21] is only specified for undirected graphs.)

Computational cost Related to the above, hopefully an example can be developed that can be simulated on available hardware in a reasonable amount of time. Whether this occurs will only become clear later.

Cost probability distributions The state space $\mathcal{S}$, and hence the observation space $\mathcal{O}$, will naturally have to contain not only the agent's current location in the graph, but also information about the costs on edges between nodes.[1] It is common, though not strictly necessary, in reinforcement learning to assume that the state space $\mathcal{S}$ and action space $\mathcal{A}$ are both finite, for mathematical tractability. To this end, it is necessary to use discrete probability distributions for the costs on each node of the graph.

Furthermore, it is clear that the costs on edges touching the same node cannot reasonably be considered independent. If traffic is heavy between nodes v and v', then that traffic presumably continues to flow

[1] In designs with more elaborate graphs, presumably the state space and observation space will also have to include information concerning the delivery origin and destination.

elsewhere in the graph, unless one of the nodes is a destination. At this point it is not entirely clear how to enforce this structure. Perhaps costs should be drawn from some sort of discrete approximation to a multivariate normal distribution, where covariances between edges decrease as the distance between those edges increases.

2.3.2 Defining the Markov decision process

Another important aspect of this mini-project is attempting to clearly define the partially observable Markov decision process. To that end, we consider each component in turn below.

State space If the graph contains n edges, then any state $s \in \mathcal{S}$ should presumably be a vector of length $n+1$, containing all current costs and also the agent's current position in the graph. If the cost distribution for the i^{th} edge has $k_i \in \mathbb{N}$ possible values, then the we have the upper bound

$$|\mathcal{S}| \leq |V| \cdot \prod_{i=1}^{n} k_i$$

on the size of the state space.

Action space It seems clear that, given any state $s \in \mathcal{S}$ (and observation $o \in \mathcal{O}$), the actions available to the agent should be movements to nodes connected to the current node. This necessitates a state-dependent action space of the form $\mathcal{A}(s)$, which is somewhat more complicated than is standard. It is clear that

$$|\mathcal{A}(s)| = \deg v,$$

where $v \in V$ is the state component of the vector s.

Transition function Transitions should presumably be deterministic, since

Observation function

Cost function The cost function (ordinarily the reward function) should be a function $c\colon \mathcal{S} \times \mathcal{A} \to \mathbb{N}$, giving the cost accrued when taking action a in state s, i.e. the cost on the edge travelled.

2.4 Predicted potential impact

3 Project Report

3.1 Project Description

3.1.1 Problems With the Proposed Project

The proposed project concerned an application of Gaussian processes on graphs to Markov decision processes, the latter being the standard framework for reinforcement learning. The general idea was that a reinforcement learning agent would be tasked with estimating shortest paths on a weighted graph, with only partially known and perhaps evolving weights.

However, it was never precisely clear how to set up the what type of Markov decision process to choose for this problem and how to set it up. For example, since the graph weights were only partially known, it might make sense to use a partially observable Markov decision process, but it was not clear how to precisely define the partially observable Markov decision process. In an email dated November 2$^{\text{nd}}$, Professor Pleiss suggested a somewhat different setup in which the costs on the graph edges are known, but the rewards on the graph vertices are unknown. This suggestion was presumably made to conform with the fact that the graph Matérn Gaussian process described in [Bor+21] is define on the graph vertices, not its edges, but this setup is no longer obviously a partially observable Markov decision process.

3.1.2 A New Project Direction: Estimating Shortest Paths

We decided to abandon the proposed connection to reinforcement learning and re-frame the project in terms of graph theory and Gaussian processes. More precisely, we consider the following problem. Suppose we have a weighted graph $G = (V, E)$ with weight function $w \colon E \to \mathbb{R}$, and only a subset of the weights are known. Mathematically, we can suppose there is some strict subset $E' \subset E$, and only the restriction $w \restriction E' \colon E' \to \mathbb{R}$ is known. We would like to estimate the entire weight function $w \colon E \to \mathbb{R}$, via a Gaussian process. This estimate of w might be written $\hat{w}$. We would like that $\hat{w} \restriction E' = w \restriction E'$, and that the values of $\hat{w}$ over the unobserved weights on edges $E \setminus E'$ are estimated via a Gaussian process.

In particular, we would like to develop estimates of the unobserved weights in order to compute *estimated shortest paths* on the graph G. Given estimates of the unobserved weights, estimated shortest paths are calculated by the application of a shortest path algorithm, for example Dijkstra's algorithm, on what we might call the graph $\hat{G}$, an edge-weighted graph with weight function $\hat{w}$.

Given this new project direction, we must consider the following issues:

1. how to formulate the estimation of w mathematically,

2. how to implement the estimation of w computationally,

3. how to evaluate the performance of a method for estimating w.

3.2 Mathematical Formulation

An initial stumbling block is that we wish to learn $w \colon E \to \mathbb{R}$, but the graph Matérn Gaussian process as defined in [Bor+21] is defined on the *vertices* of a graph, not its edges. (It would be helpful to understand why we cannot define a Gaussian process on the edges of a weighted graph, since this is would be the naïve choice.)

Given this limitation, we would like to adapt a weighted graph so that its edges become vertices, in some sense. There at at least two ways to do this, which will be described below.

3.2.1 First Method: Add Extra Vertices to G

Given a weighted graph $G_1 = (V, E)$, one possible approach to adapt G_1 appears to be to add vertices to $G1$ on the existing edges, thereby splitting each edge into two edges. A simple example is shown in Figures 2 and 3. Beginning in Figure 2 with a graph G_1, this graph is then transformed into a graph G_2 in Figure 3.

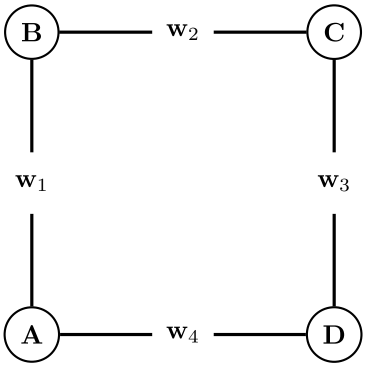

Figure 2: This figure shows a simple edge-weighted graph G_1. If only some subset of the weights $\{\mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3, \mathbf{w}_4\}$ are known, we would like to use a Gaussian process in order to estimate the unknown weights. However, because the graph Matérn Gaussian process is defined on the *vertices* of an edge-weighted graph, it is necessary to turn the edges of G_1 into nodes, in some manner.

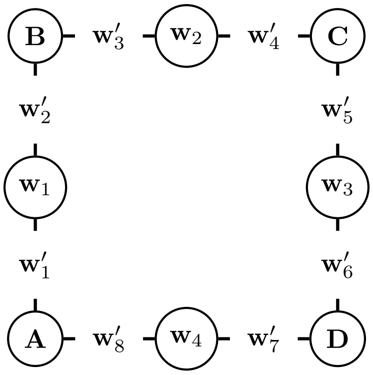

Figure 3: This figure shows one possible method to convert the graph G_1 from Figure 2 into a graph G_2 suitable for a graph Matérn Gaussian process. In graph G_2, we have inserted new vertices on each edge of G_1 and allowed each new vertex to inherit the appropriate edge. However, it is unclear how the new weights on the new, shortened vertices should be determined.

This appears to be roughly the approach followed in one of the experiments described in Appendix A of [Bor+21], as suggested by the following sentence from the aforementioned Appendix:

> We bind the traffic congestion data to the graph by adding additional nodes that subdivide existing edges of the graph at the location of the measurement points.

One apparent issue with this method is that it's not clear how to determine the edge weights for the new edges. It's also unclear how the original structure of G_1 is preserved by G_2.

3.2.2 Second Method: Convert G Into Its Line Graph $L(G)$

Informally, the line graph $L(G_1)$ of a graph G_1 is a graph such that the edges of G_1 are vertices of $L(G_1)$, while the vertices of G_1 are edges of $L(G_1)$. (This is an over-simplification.) The second approach is to form the line graph of G_1 and then apply the graph Matérn Gaussian process to $L(G_1)$. A simple example is shown in Figure 4.

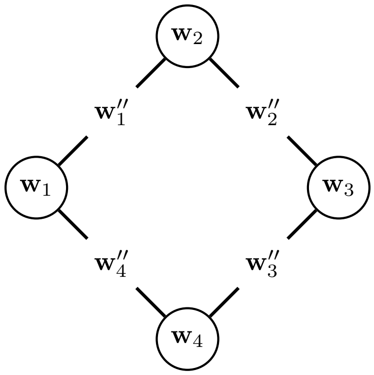

Figure 4: This figure shows another possible method to convert the graph G_1 from Figure 2 into a graph G_3 suitable for a graph Matérn Gaussian proecess. The graph G_3 is the line graph of G_1. One possible issue with this method is that for more complicated graphs, it's not clear how much of the original structure of G_1 is preserved by G_3. The graph Matérn Gaussian process will be applied on G_3, but the inferences made are ultimately only of interest in the context of the unknown edge weights on graph G_1.

For the line graph approach, similar to the first method, it is also unclear how to determine the new edge weights. It's also unclear how much of the original structure of G_1 is preserved by $L(G_1)$.

3.2.3 Discussion of Both Methods

Without much experience with graphs, it is difficult to have much intuition for the advantages and disadvantages of each of the two methods described above. Presumably, both should be explored. However, the following observations can be made:

1. Since the graph Matérn Gaussian process will be applied to either G_2 or G_3, when the actual graph of interest is G_1, presumably it is important that the graph to which the process is applied be as structurally similar to G_1 as possible, in order to maximize the accuracy of the estimates of the unknown weights of G_1. The line graph G_3 appears much more similar to G_1 than doe G_2, but this is a very simple example, and does not necessarily generalize.

2. It seems apparent that the first method, resulting in G_2, results in a graph that contains more information than graph G_3. In graph G_2, the number of nodes and weights have both doubled compared to G_1, whereas the line graph G_3 is the same size a the original graph G_1. One would suspect that this might have ramifications for the computational feasibility of the two approaches when the original graph G_1 is large.

3.2.4 Which Edge Weights Should be Initially Observed?

A question that is probably important to consider is *which* edge weights are initially known. For example, one could assume that any random subset of the edge weights could be known initially, or one could assume that only "contiguous" subsets of the edge weights are initially known.

This issue is presumably related to the discussion of computational implementation in Section 3.3. It is also closely related to potential applications, which are discussed in Section 3.7.

3.2.5 Exact Form of the Covariance Matrix

Another point of confusion in the mathematical formulation of the project is the precise form of the covariance matrix of the graph Matérn Gaussian process, given some observed data. The covariance function (and the related matrix) of a Gaussian process is perhaps its most important feature, so it is critical to have a solid understanding of this aspect of the project. (I am omitting discussion of the derivation of the graph Matérn Gaussian process itself, which is not clear either. Presumably understanding this would also be helpful.)

There is a brief discussion of the covariance matrix in Appendix B of [Bor+21], which appears to proceed as follows. Given an edge-weighted graph $G = (V, E)$ with non-negative weights, the graph Laplacian $\boldsymbol{\Delta}$ of G is a symmetric, positive semi-definite matrix admitting an eigen-decomposition $\boldsymbol{\Delta} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^\intercal$. Given some subset of vertices $\boldsymbol{x} \in V^n$, we define $\mathbf{U}(\boldsymbol{x})$ to be the $n \times d$ submatrix of $\mathbf{U}$ with rows corresponding to the elements of $\boldsymbol{x}$. Then it appears that the covariance matrix $\mathbf{K}_{\boldsymbol{xx}}$ of the graph Matérn Gaussian process is given by

$$\mathbf{K}_{\boldsymbol{xx}} = \mathbf{U}(\boldsymbol{x})\Phi(\boldsymbol{\Lambda})^{-2}\mathbf{U}^\intercal,$$

where Φ is as defined in (7). However, it is not clear how this result is derived, or whether it is the general form of a covariance matrix for a graph Matérn Gaussian process. It is similarly unclear how this result should actually be used. Is $\boldsymbol{x}$ to be interpreted as the data, i.e. the edge weights of G, that were actually observed? This is unclear.

3.3 Computational Implementation

It is not clear how training would proceed. For example, given an edge-weighted graph G where some subset of the edge weights are known, and another graph G' that is in a suitable form of the graph Matérn Gaussian process, is the prior process on G' updated to the posterior (given the observed data) one time, or is this done in a more iterative fashion? Perhaps this will depend on the characteristics of the original graph G and how the data are gathered. If the data are gathered in one shot, then perhaps we will only form the posterior process one time, but if the data are gathered incrementally, or the observed weights are changing over time, then presumably we will need to form the posterior process multiple times.

3.4 Performance Evaluation

Suppose we have an edge-weighted graph $G = (V, E)$, and we have observed the weights on some strict subset $E' \subset E$. Furthermore, suppose we have used a graph Matérn Gaussian process to estimate the values of the unobserved weights on the remaining edges $E \setminus E'$. It is important to develop some method of evaluating the quality of our weight estimations. There seem to be at least a few possible approaches.

First, we can use some sort of loss function, e.g. squared loss, by comparing the estimated weights with their true values. For example, if a graph G has N edge weights, and $1 < n < N$ of these weights are observed, then we can form the estimates $\hat{w}_{n+1}, \hat{w}_{n+2}, \ldots, \hat{w}_N$ and then compute the loss

$$\sum_{i=n+1}^{N} (w_i - \hat{w}_i)^2 . \tag{8}$$

In a somewhat similar vein, we might be interested in confidence intervals concerning our estimates of the unknown weights. It is not clear that any sort of large sample theory makes sense on a finite graph, but perhaps if the edge weights are drawn from probability distributions, then we might observe many weights over time, and perhaps large sample theory could have some application over many observations of the edge weights.

Evaluating the quality of our weight estimates via a loss function as in (8) makes sense insofar as a smaller loss presumably means our estimates are more useful, but this perspective misses the fact that our real goal is not to estimate the unobserved weights of G accurately, but rather to accurately estimate the *true shortest paths* on G accurately. From this perspective, the precise estimated weight values are not at all relevant if the shortest path estimates they lead to are identical to the true shortest paths. After all, the costs paid when traversing a path on G will reflect the true weight values, not their estimated values.

From this latter perspective, a good metric for performance evaluation would be concerned with comparing estimated shortest paths with the true shortest paths. However, it is not immediately clear how to measure the difference between an estimated shortest path from node A to node B, and the true shortest path from node A to node B. If the true shortest path from node A to node B involves the sequence of edges $(e_1, \ldots, e_n)$, and the estimated shortest path from node A to node B involves the edges $(e'_1, \ldots, e'_{n'})$, then perhaps one way to measure the difference between these paths would be with the loss

$$L(A, B) = \left(\sum_{i=1}^{n} w(e_i) - \sum_{i=1}^{n'} w(e'_i) \right)^2 .$$

In words, we compare the *true* cost accrued along the *true* shortest path from A to B, with the *true* cost accrued long the *estimated* shortest path from A to B. Perhaps then the total loss over G could be defined as

$$L_G = \sum_{\substack{A,B \in V \\ A \neq B}} L(A, B).$$

(Presumably L_G as defined above counts each path twice, since $L(A, B)$ and $L(B, A)$ will both be summands, so the criteria for summation above should be restricted to avoid such double counting.)

3.5 Issues Encountered During Attempted Implementation

A number of issues were encountered when attempting to implement some of the methods discussed above. These are listed below, not necessarily in chronological order.

The main theme in all the problems encountered below is a lack of understanding of both the theory I am attempting to implement (both Gaussian processes in general, and graph Matérn Gaussian processes in particular), as well as of the tools I tried to use for the implementation (including various Python libraries such as NetworkX, TensorFlow, GPflow, OSMnx, etc.).

1. The main tool for working with graphs in Python appears to be NetworkX, which I had no familiarity with, so learning how to create and manipulate graphs (for example, adding weights to a graph) involved a lot of trial and error.

2. I was intending to work with a dataset concerning traffic congestion, as in [Bor+21]. The graphs for traffic datasets appear to be typically acquired and manipulated using OSMnx, which according to the official documentation, should be installed with conda. However, the code supplied with the paper [Bor+21] is installed via pip, and I do not know how to work with both conda and pip. I initially began working with the code from [Bor+21] using pip, but then I tried to switch to conda to use OSMnx. However, I tried using conda to install the packages I had been using with pip, but conda was stuck on "solving environment" for about 36 hours before I decided to abort.

3. Some of the issues above may be related to the fact that I am borrowing a Windows laptop from the department, and it appears that Windows is generally not supported as well as macOS or Linux for the purposes of this project.

4. Given the above issues, I decided to pivot and look for alternative datasets of weighted graphs. After some searching, I settled on a dataset concerning a British television show [Dil21], because it seemed to avoid the use of OSMnx.

5. However, while attempting to prepare the covariance matrix, I encountered "tensor shape mismatch" errors. I am not sure if this is due to my lack of understanding of the covariance matrix, or my lack of familiarity with NetworkX, TensorFlow and GPflow.

Given the time limitations in this project, and the great temporal demands of another course I am taking this term, I was unable to actually implement a graph Matérn Gaussian process on a graph.

The issues described above are probably all rectificable, at least in principle, given enough time and perhaps enough support to become more knowledgeable about what I have been attempting to do during this project.

3.6 Related Work

Given the potential novelty of estimating shortest paths on an edge-weighted graph with partially unknown weights, it seems worthwhile to investigate whether any research has been done on this problem.

Searching Google for “graph estimate shortest path” returns almost exclusively material concerned with the standard problem of computing shortest paths on an edge-weighted graph with known weights. The only relevant result was [WFK22], which seems to be more concerned with the notion that computing the edge weights of a graph may involve non-trivial computation, and therefore introduces the use of weight estimators to compute edge weights with increasing accuracy, at the cost of increased computation time.

Searching Google for “graph shortest path weight unknown” again mostly returns material concerned with the standard shortest-path problem. The paper [Sze04] is also concerned with edge weight computation. It is assumed that edge weights are initially unknown, but that there is a method to query the edge weights, and the goal is to minimize the number of queries. The paper [TZ13] considers an undirected graph whose edge weights are random variables with unknown distributions, but is concerned with distributed computing.

Lastly, apparently the problem of finding shortest paths on graphs where the graph structure itself is not entirely known has been studied [PY91]; see also the Canadian traveller problem on Wikipedia.

In sum, it is not apparent that any prior work has explored the setting of an edge-weighted graph with only partially observed weights.

3.7 Potential Applications

The standard problem of finding shortest paths on an edge-weighted graph has many applications in navigation, logistics and computer network routing, among other areas. However, it is not immediately clear what kind of scenario involves a graph whose edge weights are only partially known. Perhaps we could imagine a robot intending to travel from one node on a graph to another while minimizing the cost accrued. The robot initially knows only some of the costs on the graph, and needs to estimate the other costs in order to plan its route. In this type of scenario, it probably makes sense that the costs are initially known only in a region of the graph near the robot, as opposed to any random subset of the graph. Perhaps as the robot travels along the graph, it observes more of the true costs on the graph’s edges, and it updates its predictions as it observes more of the graph.

It is possible that the literature regarding the Canadian traveller problem might be a source of other possible applications.

References

[BDT13] A. Banerjee, D. B. Dunson, and S. T. Tokdar. “Efficient Gaussian process regression for large datasets”. In: *Biometrika* 100.1 (2013), pp. 75–89.

[Bor+20] V. Borovitskiy et al. “Matérn Gaussian Processes on Riemannian Manifolds”. In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., 2020, pp. 12426–12437. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/92bf5e6240737e0326ea59846a83e076-Paper.pdf.

[Bor+21] V. Borovitskiy et al. “Matérn Gaussian Processes on Graphs”. In: *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*. Ed. by A. Banerjee and K. Fukumizu. Vol. 130. Proceedings of Machine Learning Research. PMLR, 13–15 Apr 2021, pp. 2593–2601. URL: https://proceedings.mlr.press/v130/borovitskiy21a.html.

[Coo16] J. D. Cook. *Measuring connectivity with graph Laplacian eigenvalues*. 2016. URL: https://www.johndcook.com/blog/2016/01/07/connectivity-graph-laplacians/.

[Cor+22] T. H. Cormen et al. *Introduction to Algorithms*. 4th ed. The MIT Press, 2022.

[Dil21] M. Dileo. *Doctor Who Dataset*. https://github.com/manuel-dileo/doctor-who-dataset. 2021.

[Fou23] C. Fourrier. *Introduction to Graph Machine Learning*. Hugging Face. 2023. URL: https://huggingface.co/blog/intro-graphml.

[Gar23] R. Garnett. *Bayesian Optimization*. Cambridge University Press, 2023.

[GL13] G. H. Golub and C. F. V. Loan. *Matrix Computations*. 4th ed. The Johns Hopkins University Press, 2013.

[Gri18] G. Grimmett. *Probability on Graphs: Random Processes on Graphs and Lattices*. 2nd ed. Institute of Mathematical Statistics Textbooks 8. Cambridge University Press, 2018.

[Kal21] O. Kallenberg. *Foundations of Modern Probability*. 3rd ed. Probability Theory and Stochastic Modelling 99. Springer Cham, 2021.

[KL02] R. I. Kondor and J. Lafferty. “Diffusion kernels on graphs and other discrete structures”. In: *Proceedings of the 19th international conference on machine learning*. Vol. 2002. 2002, pp. 315–322.

[Les23] J. Leskovec. *CS224W: Machine Learning with Graphs*. Stanford University. 2023. URL: https://web.stanford.edu/class/cs224w/.

[Liu+20] H. Liu et al. “When Gaussian process meets big data: A review of scalable GPs”. In: *IEEE transactions on neural networks and learning systems* 31.11 (2020), pp. 4405–4423.

[PY91] C. H. Papadimitriou and M. Yannakakis. “Shortest paths without a map”. In: *Theoretical Computer Science* 84.1 (1991), pp. 127–150.

[RW06] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2006.

[Sze04] C. Szepesvári. “Shortest path discovery problems: A framework, algorithms and experimental results”. In: *AAAI*. 2004, pp. 550–555.

[TZ13] P. Tehrani and Q. Zhao. “Distributed online learning of the shortest path under unknown random edge weights”. In: *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE. 2013, pp. 3138–3142.

[Wan+18] Z. Wang et al. “Batched large-scale Bayesian optimization in high-dimensional spaces”. In: *International Conference on Artificial Intelligence and Statistics*. PMLR. 2018, pp. 745–754.

[WFK22] E. Weiss, A. Felner, and G. A. Kaminka. “A Generalization of the Shortest Path Problem to Graphs with Multiple Edge-Cost Estimates”. In: *arXiv preprint arXiv:2208.11489* (2022).

[Whi63] P. Whittle. “Stochastic-processes in several dimensions”. In: *Bulletin of the International Statistical Institute* 40.2 (1963), pp. 974–994.