Network Structures, Concurrency, and Interpretability: Lessons from the Development of an AI Enabled Graph Database System

Hal James Cooper

Submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy
in the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2020

ABSTRACT

Network Structures, Concurrency, and Interpretability: Lessons from the Development of an AI Enabled Graph Database System

Hal James Cooper

This thesis describes the development of the SmartGraph, an AI enabled graph database. The need for such a system has been independently recognized in the isolated fields of graph databases, graph computing, and computational graph deep learning systems, such as TensorFlow. Though prior works have investigated some relationships between these fields, we believe that the SmartGraph is the first system designed from conception to incorporate the most significant and useful characteristics of each. Examples include the ability to store graph structured data, run analytics natively on this data, and run gradient descent algorithms. It is the synergistic aspects of combining these fields that provide the most novel results presented in this dissertation. Key among them is how the notion of "graph querying" as used in graph databases can be used to solve a problem that has plagued deep learning systems since their inception; rather than attempting to embed graph structured datasets into restrictive vector spaces, we instead allow the deep learning functionality of the system to natively perform graph querying in memory during optimization as a way of interpreting (and learning) the graph. This results in a concept of natural and interpretable processing of graph structured data.

Graph computing systems have traditionally used distributed computing across multiple compute nodes (e.g. separate machines connected via Ethernet or internet) to deal with large-scale datasets whilst working sequentially on problems over entire datasets. In this dissertation, we outline a distributed graph computing methodology that facilitates all the above capabilities (even in an environment consisting of a single physical machine) while allowing for a workflow more typical of a graph database than a graph computing system; massive concurrent access allowing for arbitrarily asynchronous execution of queries and analytics across the entire system. Further, we demonstrate how this methodology is key to the artificial intelligence capabilities of the system.

Contents

List of Figures

List of Tables

Acknowledgements

I offer my most profound thanks to my advisor, Professor Garud Iyengar. Despite having many duties of his own, Professor Iyengar always made time for me and my research. An accomplished scholar on a variety of topics, Professor Iyengar could always suggest promising directions and ideas from areas beyond my ken. This is but one of many ways that he helped me grow tremendously as a scholar, as I incorporated new knowledge and these new fields into my own knowledge base. As someone who also had an electrical engineering background, Professor Iyengar understood my desire for a systems thesis. He guided me on a path in my research that allowed me to accomplish this goal with the aid of the most relevant and powerful tools that the Operations Research field had to offer, and to then create my own. I am most grateful for Professor Iyengar for believing in and encouraging me and my research, even in moments when I had my own doubts.

One of the most atypical decisions I made as a doctoral candidate was to pursue internships during each summer of my studies. Through these internships at Goldman Sachs, IBM, and Graphen, Inc., I met many influential, wonderful researchers and domain experts. Through conversations with the likes of Danny Yeh and Toyotaro Suzumura, I was able to understand how academic ideas can readily translate into practical applications in industry, and took many such considerations into account when performing my research and designing my system. Most influential of these is Professor Ching-Yung Lin of Columbia, IBM, and Graphen, Inc., who has honored me by agreeing to part of my dissertation committee. It was through working with Ching-Yung and his team (and then company) that I discovered how incredibly interesting graph structured datasets could be, providing me the impetus for the direction of this dissertation.

During my studies, I had the great pleasure of working with students and professors both inside and outside the department on a variety of projects. To my one-time fellow doctoral candidates Francois Fagan and Min-Hwan Oh of the Industrial Engineering and Operations Research depart-

ment, I give my sincerest gratitude for your always kind attitudes, and dedication to our combined projects. Of the Statistics department, I thank Gonzalo Mena, Scott Linderman, Professor John Cunningham, and Professor Liam Paninski for always interesting classes, project collaborations, and research group participation.

I would like to thank Emeritus Laureate Professor Graham Goodwin of the University of Newcastle, Australia. His guidance through both summer internships (at the Priority Research Centre for Complex Dynamic Systems and Control) and my undergraduate Honours Project began my journey as a scholar, and I was very fortunate to have learned from someone of such renown and wisdom.

I'd also like to acknowledge the tireless work of the department staff of the Industrial Engineering and Operations Research Department. In particular, I would like to give thanks to Lizbeth Morales for always being ready to answer a question, Yosimir Acosta for setting up the departments computing infrastructure as I required (no matter how obscure), and Kristen Maynor for greeting me with genuine friendship from day one.

Of course, much can be said for the professors of the Industrial Engineering and Operations Research department. Through their instructive courses and consultations, I learned what was required to tackle a task the magnitude of a PhD. I am especially grateful to Professor Daniel Bienstock, who is serving as the Chair of Examination for my dissertation, as well as Professors Jaychandran Sethuraman and Ali Hirsa, who also graciously agreed to be part of my committee. I am honored to present my doctoral dissertation to scholars of such caliber.

To Lyla

Introduction

The motivating problem for this dissertation is how to perform machine learning with graph structured datasets. In existing approaches, graph structured datasets are compressed, embedded, or approximated as tensor valued objects. The rationale for this approach is that it allows existing machine learning methodologies to be readily applied to complex network and graph structured data. However, such approaches are inherently lacking in that graph structured datasets are much more complex than tensor structured data. Any tensor-based approximation of graph structure that performs well for a particular problem may turn out to be completely inadequate when that same approximation is applied to another problem.

We therefore seek a solution to the problem of graph inputs, where graphs can be explored and interpreted by machine learning models in a "natural" graph theoretic manner; by following, exploring, and understanding connections between data (in much the same way a human might), and by summarizing this information through interpretable querying procedures. This allows machine learning algorithms to explore and learn from (if necessary) whole graph structures. This approach directs the algorithm to learn the relevant topological and property structure of graphs required to solve a given graph input-based machine learning problem. As this exploration of the graph is directed by the system (and human domain expertise, if it is beneficial), this is an approach that can be applied to a wide range of problems.

One consequence of performing machine learning on graph structured datasets is that we need a system capable of dealing with the potentially enormous size of the datasets. Unlike tensor-valued inputs which may be easily batched for use in SGD methods, the size of the relevant graph structured data of a graph-input machine learning problem may not be known apriori. Furthermore, the method of performing learning and algorithm execution concurrently is not immediately obvious.

We design and implement the SmartGraph, a graph database that is inherently capable of machine learning algorithm execution on graph structured datasets in a massively concurrent fashion.

This dissertation describes the steps relevant to the research, design, and implementation of the SmartGraph. This work is an attempt at bring together ideas from the fields of graph databases, graph computing, and deep learning systems - fields that have largely developed independently - it will become clear through reading the dissertation that these fields have much more in common than previously explored. We first present the projects involving work primarily focused in the above mentioned individual research areas. These projects served as the guide for the development of the SmartGraph system.

Where possible, we indicate and allude to concepts developed later in the dissertation in order to demonstrate how these earlier projects inspired us to develop a system that either used techniques from these earlier papers in a novel manner, or demonstrated clear problems with typical approaches that needed to be acknowledged and addressed in the SmartGraph system.

In Chapter 1, we introduce methods for approximating discrete optimization problems as continuous optimization problems that can then be solved with gradient descent methods. Our particular contribution is to demonstrate how methods primarily used to optimize functions involving categorical variables can be adapted and extended into problems involving *permutation* variables [1]. Permutations are particularly important in the context of network structured data, as they can represent unlabeled adjacency matrix structures (i.e. where we know the connectome or the connections between vertices, but do not know the identity of the vertices). As many of the actions taken in the natural reading of a graph are from a discrete set (e.g. which type of edge to follow from a vertex in a relevant query), the ability to cast these settings into a continuous form where gradient descent can be used is of critical importance later in the dissertation.

Chapter 2 outlines how typical machine learning methodologies can be used to solve problems in a graph context. We show how GANs can be used to learn the important topological features of graphs in contexts where there is only *one* graph, such as a bank's complete financial transaction network, and we need to understand this single instance [2].

Though both these chapters explicitly relate to network structure, they share a fundamental approach in adapting the problem to fit the available tools that prevent them from being more widely applicable to general graph problems in machine learning.

In Chapter 3, we begin to flip this paradigm around. Instead of fitting the problem to existing tools, we begin expanding existing tools to be capable of solving the problem [3, 4]. In particular, we demonstrate how the types of techniques explored in Chapter 1 can scale problems in the decision support system sphere using Influence Diagrams (IDs), while existing advantages like interpretability (lost in the embedding techniques of Chapter 2) and efficiency, and adding new concepts like "targeted robustness", which allows us to easily introduce different types of robustness at a very fine level of detail.

We begin the description of the SmartGraph with Chapter 4, which outlines the basic "router-like" structure that is key to the distributed and concurrent functionality of the system, and gives the SmartGraph its AI capabilities. As a particular example, we focus on how the router structure allows us to solve an exponentially complex combinatorial problem related to the choice of router representations (i.e. data structures for subgraph storage) as a series of simple linear optimization problems that can be solved very quickly once each router "learns" about its own subgraph, and the connections it has to other routers [5].

In the final chapter, Chapter 5, we introduce the most significant and novel capability of the SmartGraph: the ability to perform deep learning on graph structures. The reading of the graph as an input occurs *during each iteration* of machine learning problem. This occurs as part of a querying procedure guided by both the user and gradient descent algorithms. That is, we read the graph as an input into our algorithms not by summarizing the graph into factors or a vector-approximation beforehand, but by exploring the graph during execution. As a particular use case, we demonstrate how this approach can be used for product recommendation on graph data sets [6] where there is some notion of interaction between groups of users, and between a user and the products they own.

We then conclude the dissertation by highlighting promising areas of future research related to the philosophy of the overall system design (where we deal with such topics for individual projects within the corresponding chapters).

Chapter 1

Optimization with Continuous Approximations to Discrete Distributions

This chapter introduces methods for approximating discrete distributions with continuous distributions for certain classes of objects. Our key contribution in this chapter is to extend the approach to a new class of object, the permutation matrix. In the context of this dissertation, the entire chapter is relevant due to its importance in allowing us to optimize over discrete actions taken while processing a graph input.

1.1 Variational Inference and the Reparameterization Trick

Given a model with data y, likelihood $p(y \mid x)$, and prior $p(x)$, variational Bayesian inference algorithms aim to approximate the posterior distribution $p(x \mid y)$ with a more tractable distribution $q(x; \theta)$, where "tractable" means that one can sample from q and evaluate it pointwise (including its normalization constant) [7]. We find this approximate distribution by searching for the parameters θ that minimize the Kullback-Leibler (KL) divergence between q and the true posterior, or equivalently, maximize the evidence lower bound (ELBO),

$$\mathcal{L}(\theta) \triangleq \mathbb{E}_q \left[\log p(x, y) - \log q(x; \theta) \right].$$

Perhaps the simplest method of optimizing the ELBO is stochastic gradient ascent. However, computing $\nabla_\theta \mathcal{L}(\theta)$ requires some care since the ELBO contains an expectation with respect to a distribution that depends on these parameters.

When x is a continuous random variable, one can sometimes leverage the *reparameterization trick* [8, 9]. Specifically, in some cases we can simulate from q via the following equivalence,

$$x \sim q(x;\theta) \iff z \sim r(z), \quad x = g(z;\theta),$$

where r is a distribution on the "noise" z and where $g(z;\theta)$ is a deterministic and differentiable function. The reparameterization trick effectively "factors out" the randomness of q. With this transformation, we can bring the gradient inside the expectation as follows,

$$\nabla_\theta \mathcal{L}(\theta) = \mathbb{E}_{r(z)}\Big[\nabla_\theta \log p(g(z;\theta) \,|\, y) - \nabla_\theta \log q(g(z;\theta);\theta)\Big]. \tag{1.1}$$

This gradient can be estimated by Monte Carlo Simulation, and in practice, leads to lower variance estimates of the gradient as compared to, for example, the REINFORCE estimator [10, 11].

However, the gradient in (1.1) can only be computed if x is continuous. Concurrently, [12] and [13] proposed the "Concrete distribution" (or "Gumbel-softmax" method) for discrete variational inference. It is based on the following observation: discrete probability mass functions $q(x;\theta)$ can be seen as densities with atoms on the vertices of the simplex; i.e. on the set of *one-hot* vectors $\{e_n\}_{n=1}^N$, where $e_n = (0, 0, \ldots, 1, \ldots, 0)^\mathsf{T}$ is a length-N binary vector with a single 1 in the n-th position. This motivates a natural relaxation: let $q(x;\theta)$ be a density on the interior of the simplex instead, and anneal this density such that it converges to an atomic density on the vertices. Figure 1.1a illustrates this idea. Below we discuss some the various methods used for the purposes of inference and optimization regarding discrete variables, and provide more detail on the Concrete distribution.

1.1.1 The Reinforce Estimator

The REINFORCE estimator [14], also known as the "score function estimator" is probably the most-well known estimator for use in optimizing discrete variables, and is unsurprisingly the basis of many reinforcement learning methods. It uses the result $\nabla_\theta \mathbb{E}_{p(b|\theta)}[f(b)] \approx \mathbb{E}_p[f(b)\dfrac{\partial}{\partial\theta}\log p(b|\theta)]$

for $b \sim p(b|\theta)$ [15]. While this estimator is unbiased, it typically has very high variance (therefore requiring many samples to get a good estimate of the gradient). As will be discussed in Chapter 5, we require the system developed in this dissertation to use a very small a number of samples since each "sample" involves a full graph query, potentially of very high computational cost. Thus, the REINFORCE approach is not suited for our applications.

1.1.2 The Concrete Distribution and Extensions

Viewing x as a vertex of the simplex motivates a natural relaxation: rather than restricting ourselves to atomic measures, consider continuous densities on the simplex. Suppose the density of x is defined by the transformation,

$$\begin{aligned} z_n &\overset{\text{iid}}{\sim} \text{Gumbel}(0,1) \\ \psi_n &= \log \alpha_n + z_n \\ x &= \mathsf{softmax}(\psi/\lambda) \\ &= \left(\frac{e^{\psi_1/\lambda}}{\sum_{n=1}^N e^{\psi_n/\lambda}}, \ldots, \frac{e^{\psi_N/\lambda}}{\sum_{n=1}^N e^{\psi_n/\lambda}} \right). \end{aligned} \tag{1.2}$$

The output x is now a point on the simplex, and the parameter $\alpha = (\alpha_1, \ldots, \alpha_N) \in \mathbb{R}_+^N \subseteq \theta$ can be optimized via stochastic gradient ascent with the reparameterization trick.

The Gumbel distribution leads to a nicely interpretable model: adding i.i.d. Gumbel noise to $\log \alpha$ and taking the argmax yields an exact sample from the normalized probability mass function $\bar{\alpha}$, where $\bar{\alpha}_n = \alpha_n / \sum_{m=1}^N \alpha_m$ [16]. The softmax is a natural relaxation. As the temperature λ goes to zero, the softmax converges to the argmax function.

It should be noted that the Concrete distribution is a biased estimator of the gradient, though it has much lower variance than REINFORCE (consider how in comparison to the REINFORCE estimator, the Concrete distribution has more "knowledge" of how a function is constructed with respect to the parameters θ). Despite this bias, it tends to converge appropriately. More recent works, such as REBAR [17] and RELAX [15] attempt to lower the variance further (e.g. through control variates [18]) and create an unbiased estimate; however, this comes at the cost of a considerable increase in implementation complexity and iteration run-times. Due to the simplicity of the standard Concrete distribution and its success despite its bias, the works in this dissertation

(in Chapter 3 and Chapter 5), we will build upon the standard Concrete distribution rather than its "successors".

In this chapter, we investigate and develop alternate methods of continuous approximations to discrete systems for the purpose of optimization. This is not because we are attempting to reduce variance or bias, but rather because the methods mentioned, thus far, are impractical for the much more complex discrete system of focus in this chapter, namely the permutation. Due to the resulting implementation complexity, the methods developed for this purpose are self-contained in this chapter. Future work may involve the addition of these approaches to the SmartGraph system, though results in later chapters suggest it likely unnecessary. Following the main results of this chapter, we outline how the approaches developed here can be used for the more common categorical case that is actively used in the rest of this work.

1.2 The Permutation Problem

Many matching, tracking, sorting, and ranking problems require probabilistic reasoning about possible permutations, a set that grows factorially with the size of the number of inputs. Combinatorial optimization algorithms may enable efficient point estimation, but fully Bayesian inference poses a severe challenge in this high-dimensional, discrete space. To surmount this challenge, we start by relaxing the discrete set of permutation matrices to its convex hull, the Birkhoff polytope, i.e. the set of doubly-stochastic matrices. We then introduce two novel transformations: an invertible and differentiable stick-breaking procedure that maps an unconstrained space to the Birkhoff polytope, and a map that rounds points toward the vertices of the polytope [1]. Both transformations include a temperature parameter that, in the limit, concentrates the densities on permutation matrices. We exploit these transformations and reparameterization gradients to introduce variational inference over permutation matrices, and we demonstrate its utility in a series of experiments.

1.3 Introduction to Permutation Inference

Permutation inference is central to many modern machine learning problems. Identity management [19] and multiple-object tracking [20, 21] are fundamentally concerned with finding a permutation that maps an observed set of items to a set of canonical labels. Ranking problems, critical to search

and recommendation systems, require inference over the space of item orderings [22–24]. Furthermore, as many probabilistic models, like preferential attachment network models [25] and repulsive point process models [26], incorporate a latent permutation into their generative processes; inference over model parameters requires integrating over the set of permutations that could have given rise to the observed data. In neuroscience, experimentalists now measure whole-brain recordings in *C. Elegans* [27, 28], a model organism with a known synaptic network [29]; a current challenge is matching the observed neurons to corresponding nodes in the reference network. In Section 1.5.2, we address this problem from a Bayesian perspective in which permutation inference is a central component of a larger inference problem involving unknown model parameters and hierarchical structure.

The task of computing optimal point estimates of permutations under various loss functions has been well studied in the combinatorial optimization literature [30–32]. However, many probabilistic tasks, like the neural identity inference problem, require reasoning about the posterior distribution over permutation matrices. A variety of Bayesian permutation inference algorithms have been proposed, leveraging sampling methods [33–35], Fourier representations [21, 36], as well as convex [37] and continuous [38] relaxations for approximating the posterior distribution. We propose an alternative, leveraging stochastic variational inference [39] and reparameterization gradients [9, 40] to derive a scalable and efficient permutation inference algorithm.

Section 1.3.1 introduces relevant definitions, and Section 1.3.2 outlines prior work on permutation inference, variational inference, and continuous relaxations. Section 1.4 presents our primary contribution: a pair of transformations that enable variational inference over doubly-stochastic matrices, and, in the zero-temperature limit, permutations, via stochastic variational inference. In the process, we show how these transformations connect to recent work on discrete variational inference [12, 13]. Sections 1.5.1 and 1.5.2 present a variety of experiments that illustrate the benefits of the proposed variational approach.

1.3.1 Definitions and Notation

A permutation is a bijective mapping of a set onto itself. When this set is finite, the mapping is conveniently represented as a binary matrix $X \in \{0,1\}^{N\times N}$ where entry $x_{m,n} = 1$ implies that element m is mapped to element n. Since permutations are bijections, both the rows and columns

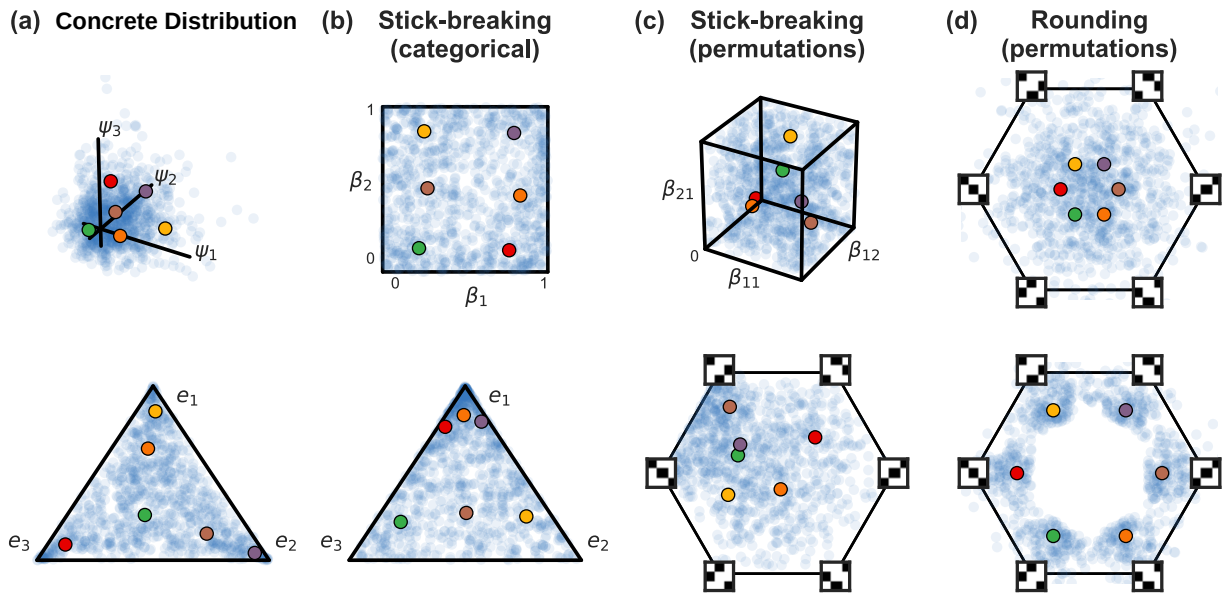


Figure 1.1: Reparameterizations of discrete polytopes. From left to right: (a) The Gumbel-softmax, or "Concrete" transformation maps Gumbel r.v.'s $\psi \in \mathbb{R}^N$ (blue dots) to points in the simplex $x \in \Delta_N$ by applying the softmax. Colored dots are random variates that aid in visualizing the transformation. (b) Stick-breaking offers an alternative transformation for categorical inference, here from points $\beta \in [0,1]^{N-1}$ to Δ_N, but the ordering of the stick-breaking induces an asymmetry in the transformation. (c) We extend this stick-breaking transformation to reparameterize the Birkhoff polytope, i.e. the set of doubly-stochastic matrices. We show how $\mathcal{B}_3$ is reparameterized in terms of matrices $B \in [0,1]^{2\times 2}$ These points are mapped to doubly-stochastic matrices, which we have projected onto $\mathbb{R}^2$ below (stencils show permutation matrices at the vertices). (d) Finally, we derive a "rounding" transformation that moves points in $\mathbb{R}^{N\times N}$ nearer to the closest permutation matrix, which is found with the Hungarian algorithm. This is more symmetric, but does not map strictly onto $\mathcal{B}_N$.

of X must sum to one. From a geometric perspective, the Birkhoff-von Neumann theorem states that the convex hull of the set of permutation matrices is the set of doubly-stochastic matrices; i.e. non-negative square matrices whose rows and columns sum to one. The set is called the *Birkhoff polytope*. Let $\mathcal{B}_N$ denote the Birkhoff polytope of $N \times N$ doubly-stochastic matrices. The row- and column-normalization constraints restrict $\mathcal{B}_N$ to a $(N-1)^2$ dimensional subset of $\mathbb{R}^{N\times N}$. Despite these constraints, we have a number of efficient algorithms for working with these objects. The *Sinkhorn-Knopp algorithm* [41] maps the positive orthant onto $\mathcal{B}_N$ by iteratively normalizing the rows and columns, and the *Hungarian algorithm* [30, 31] solves the minimum cost bipartite matching problem, optimizing a linear objective over the set of permutation matrices in cubic time.

1.3.2 Related Work

A number of previous works have considered approximate methods of posterior inference over the space of permutations. When a point estimate will not suffice, sampling methods like Markov chain Monte Carlo (MCMC) algorithms may yield a reasonable approximate posterior for simple problems [33]. In [35] an importance sampling algorithm was developed that fills in count matrices one row at a time, showing promising results for matrices with $O(100)$ rows and columns. In [42], the authors considered using the Hungarian algorithm within a Perturb-and-MAP algorithm for approximate sampling. Another line of work considered inference in the spectral domain, approximating distributions over permutations with the low-end frequency Fourier components [21, 36]. Perhaps most relevant to this work, [38] proposes a continuous relaxation from permutation matrices to points on a hypersphere, and then uses the von Mises-Fisher (vMF) distribution to model distributions on the sphere's surface. Finally, ranking problems are a special case of a matching problem in which the labels are the ordered set of integers $\{1, \ldots, N\}$. The Placket-Luce model is one model for rankings that is parameterized by a "score" for each item, and it admits efficient Bayesian inference algorithms [43]. In general matching problems, however, the output is not ordered, and we instead need scores for each item-label mapping. The methods presented here address general matching problems.

1.4 Variational Permutation Inference via Reparameterization

The Concrete distribution scales linearly with the support, rendering it prohibitively expensive for direct use on the set of $N!$ permutations. Instead, we develop two transformations to map $O(N^2)$-dimensional random variates to points in or near the Birkhoff polytope. Like the Concrete distribution, these transformations will be controlled by a temperature that concentrates the resulting density near permutation matrices. The first method is a novel "stick-breaking" construction; the second rounds points toward permutations with the Hungarian algorithm. We present these in turn and then discuss their relative merits.

1.4.1 Stick-Breaking Transformations to the Birkhoff Polytope

Stick-breaking is well-known as a construction for the Dirichlet process [44]; here we show how the same intuition can be extended to more complex discrete objects. Let B be a matrix in $[0,1]^{(N-1)\times(N-1)}$; we will transform it into a doubly-stochastic matrix $X \in [0,1]^{N\times N}$ by filling in entry by entry, starting in the top left and raster scanning left to right then top to bottom. Denote the (m,n)-th entries of B and X by β_{mn} and x_{mn}, respectively.

Each row and column has an associated unit-length "stick" that we allot to its entries. The first entry in the matrix is given by $x_{11} = \beta_{11}$. As we work left to right in the first row, the remaining stick length decreases as we add new entries. This reflects the row normalization constraints. The first row follows the standard stick-breaking construction,

$$x_{1n} = \beta_{1n}\left(1 - \sum_{k=1}^{n-1} x_{1k}\right) \qquad \text{for } n = 2,\ldots,N-1$$
$$x_{1N} = 1 - \sum_{n=1}^{N-1} x_{1n}.$$

This is illustrated in Figure 1.1b, where points in the unit square map to points in the simplex. Here, the blue dots are two-dimensional $\mathcal{N}(0, 4I)$ variates mapped through a coordinate-wise logistic function.

Subsequent rows are more interesting, requiring a novel advance on the typical uses of stick breaking. Here we need to conform to row and column sums (which introduce upper bounds), and a lower bound induced by stick remainders that must allow completion of subsequent sum constraints. Specifically, the remaining rows must now conform to both row- and column-constraints. That is,

$$x_{mn} \le 1 - \sum_{k=1}^{n-1} x_{mk} \qquad \text{(row sum)}$$
$$x_{mn} \le 1 - \sum_{k=1}^{m-1} x_{kn} \qquad \text{(column sum)}.$$

Moreover, there is also a lower bound on x_{mn}. This entry must claim enough of the stick such that what remains fits within the confines imposed by subsequent column sums. That is, each column sum places an upper bound on the amount that may be attributed to any subsequent entry. If the remaining stick exceeds the sum of these upper bounds, the matrix will not be doubly-stochastic.

Thus,

$$\underbrace{1 - \sum_{k=1}^{n} x_{mk}}_{\text{remaining stick}} \leq \underbrace{\sum_{j=n+1}^{N} (1 - \sum_{k=1}^{m-1} x_{kj})}_{\text{remaining upper bounds}}.$$

Rearranging terms, we have,

$$x_{mn} \geq 1 - N + n - \sum_{k=1}^{n-1} x_{mk} + \sum_{k=1}^{m-1} \sum_{j=n+1}^{N} x_{kj}.$$

Of course, this bound is only relevant if the right hand side is greater than zero. Taken together, we have $\ell_{mn} \leq x_{mn} \leq u_{mn}$, where,

$$\ell_{mn} \triangleq \max \left\{ 0,\, 1 - N + n - \sum_{k=1}^{n-1} x_{mk} + \sum_{k=1}^{m-1} \sum_{j=n+1}^{N} x_{kj} \right\}$$
$$u_{mn} \triangleq \min \left\{ 1 - \sum_{k=1}^{n-1} x_{mk},\, 1 - \sum_{k=1}^{m-1} x_{kn} \right\}.$$

Accordingly, we define $x_{mn} = \ell_{mn} + \beta_{mn}(u_{mn} - \ell_{mn})$. The inverse transformation from X to B is analogous. We start by computing z_{11} and then progressively compute upper and lower bounds and set $\beta_{mn} = (x_{mn} - \ell_{mn})/(u_{mn} - \ell_{mn})$.

To complete the reparameterization, we define a parametric, temperature-controlled density from a standard Gaussian matrix $Z \in \mathbb{R}^{(N-1)\times(N-1)}$ to the unit-hypercube B. Let,

$$\psi_{mn} = \mu_{mn} + \nu_{mn} z_{mn},$$
$$\beta_{mn} = \sigma\left(\psi_{mn}/\lambda\right),$$

where $\theta = \{\mu_{mn}, \nu_{mn}^2\}_{m,n=1}^{N}$ are the mean and variance parameters of the intermediate Gaussian matrix Ψ, $\sigma(u) = (1 + e^{-u})^{-1}$ is the logistic function, and λ is a temperature parameter. As $\lambda \to 0$, the values of β_{mn} are pushed to either zero or one, depending on whether the input to the logistic function is negative or positive, respectively. As a result, the doubly-stochastic output matrix X is pushed toward the extreme points of the Birkhoff polytope, the permutation matrices. This map is illustrated in Figure 1.1c for permutations of $N = 3$ elements. Here, the blue dots are samples

of B with $\mu_{mn} = 0$, $\nu_{mn} = 2$, and $\lambda = 1$.

We compute gradients of this transformation with automatic differentiation. Since this transformation is "feed-forward," its Jacobian is lower triangular. The determinant of the Jacobian, necessary for evaluating the density $q_\lambda(X;\theta)$, is a simple function of the upper and lower bounds. While this map is peculiar in its reliance on an ordering of the elements, as discussed in Section 1.4.3, it is a novel transformation to the Birkhoff polytope that supports variational inference.

1.4.2 Rounding Toward Permutation Matrices

While relaxing permutations to the Birkhoff polytope is intuitively appealing, it is not strictly required in order to have a differentiable relaxation that can be used to generate samples. We can simply get "near" the set, and use such samples as an approximation to a truly doubly stochastic matrix. In addition, using stick-breaking introduces a bias due to the ordering of our stick-breaking. We are thus motivated to find a relaxation that is more "balanced" across permutations. For example, consider the following procedure for sampling a point near the Birkhoff polytope:

(i) Input $Z \in \mathbb{R}^{N\times N}$, $M \in \mathbb{R}_+^{N\times N}$, and $V \in \mathbb{R}_+^{N\times N}$;

(ii) Map $M \to \widetilde{M}$, a point in the Birkhoff polytope, using the Sinkhorn-Knopp algorithm;

(iii) Set $\Psi = \widetilde{M} + V \odot Z$ where $\odot$ denotes elementwise multiplication;

(iv) Find $\mathsf{round}(\Psi)$, the nearest permutation matrix to Ψ, using the Hungarian algorithm;

(v) Output $X = \lambda\Psi + (1-\lambda)\mathsf{round}(\Psi)$.

This procedure defines a mapping $X = g_\lambda(Z;\theta)$ with $\theta = \{M, V\}$. When the elements of Z are independently sampled from a standard normal distribution, it implicitly defines a distribution over matrices X parameterized by θ. Furthermore, as λ goes to zero, the density concentrates on permutation matrices. A simple example is shown in Figure 1.1d, where $M = \frac{1}{N}\mathbf{1}\mathbf{1}^\mathsf{T}$ with $\mathbf{1}$ a vector of all ones, $V = 0.4^2\mathbf{1}\mathbf{1}^\mathsf{T}$, and $\lambda = 0.5$. We use this procedure to define a variational distribution with density $q_\lambda(X;\theta)$.

To compute the ELBO and its gradient in (1.1), we need to evaluate $q_\lambda(X;\theta)$. By construction, steps (i) and (ii) involve differentiable transformations of parameter M to set the mean close to the Birkhoff polytope, but since these do not influence the distribution of Z, the non-invertibility of the Sinkhorn-Knopp algorithm poses no problems. Had we applied this algorithm directly to Z, this

would not be true. The challenge in computing the density stems from the rounding in steps (iv) and (v).

To compute $q_\lambda(X;\theta)$, we need the inverse $g_\lambda^{-1}(X;\theta)$ and its Jacobian. The inverse is straightforward: when $\lambda \in [0,1)$, $\mathsf{round}(\Psi)$ outputs a point strictly closer to the nearest permutation, implying $\mathsf{round}(\Psi) \equiv \mathsf{round}(X)$. Thus, the inverse is $g_\lambda^{-1}(X;\theta) = \left(\frac{1}{\lambda}X - \frac{1-\lambda}{\lambda}\mathsf{round}(X) - \widetilde{M}\right) \oslash V$, where $\oslash$ denotes elementwise division. A slight wrinkle arises from the fact that step (v) maps to a subset $\mathcal{X}_\lambda \subset \mathbb{R}^{N\times N}$ that excludes the center of the Birkhoff polytope (note the "hole" in Figure 1.1d), but this inverse is valid for all X in that subset.

The Jacobian is more challenging due to the non-differentiability of round. However, since the nearest permutation output only changes at points that are equidistant from two or more permutation matrices, round is a piecewise constant function with discontinuities only at a set of points with zero measure. Thus, the change of variables theorem still applies.

With the inverse and its Jacobian, we have

$$q_\lambda(X;\theta) = \prod_{m=1}^{N}\prod_{n=1}^{N} \frac{1}{\lambda\nu_{mn}} \mathcal{N}\left(z_{mn};\, 0, 1\right) \times \mathbb{I}[X \in \mathcal{X}_\lambda],$$

where $z_{mn} = [g_\lambda^{-1}(X;\theta)]_{mn}$ and ν_{mn} are the entries of V. In the zero-temperature limit we recover a discrete distribution on permutation matrices; otherwise the density concentrates near the vertices as $\lambda \to 0$. This transformation leverages computationally efficient algorithms like Sinkhorn-Knopp and the Hungarian algorithm to define a temperature-controlled variational distribution near the Birkhoff polytope, and it enjoys many theoretical and practical benefits.

1.4.3 Theoretical Considerations

Continuous relaxations require re-thinking the objective: the model log-probability is defined with discrete latent variables, but our relaxed posterior is a continuous density. As in [12], we instead maximize a relaxed ELBO. We assume the functional form of the likelihood remains unchanged, and simply accepts continuous values instead of discrete. However, we need to specify a new continuous prior $p(X)$ over the relaxed discrete latent variables, here, over relaxations of permutation matrices. It is important that the prior be sensible: ideally, the prior should penalize values of X that are far from permutation matrices.

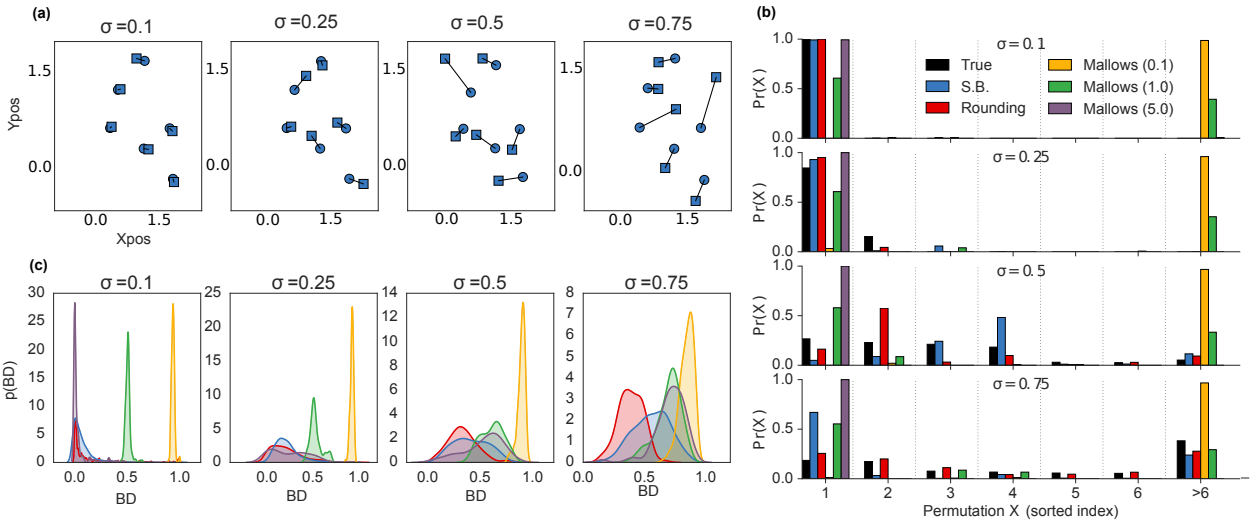


Figure 1.2: Synthetic matching experiment results. The goal is to infer the lines that match squares to circles. (a) Examples of center locations (circles) and noisy samples (squares), at different noise variances. (b) For illustration, we show the true and inferred probability mass functions for different method (rows) along with the Battacharya distance (BD) between them for a selected case of each σ (columns). Permutations (indices) are sorted from the highest to lowest actual posterior probability. Only the 10 most likely configurations are shown, and the 11th bar represents the mass of all remaining configurations. (c) KDE plots of Battacharya distances for each parameter configuration (based on 200 experiment repetitions) for each method and parameter configuration. For comparison, stick-breaking, rounding, and Mallows ($\theta = 1.0$) have BD's of .36, .35, and .66, respectively, in the $\sigma = 0.5$ row of (b).

For a categorical distribution, we can define a prior that uses a mixture of Gaussians around each vertex, $p(x) = \frac{1}{N}\sum_{n=1}^{N}\mathcal{N}(x \,|\, e_k, \eta^2)$ (indeed, this is the prior used later on in Section 1.7). This can be extended to permutations, where we use a mixture of Gaussians for each coordinate. Although this prior puts significant mass around invalid points (e.g. $(1, 1, \ldots, 1)$), it penalizes X that are far from $\mathcal{B}_N$. The prior formulation is given in (1.3).

$$p(X) = \prod_{m=1}^{N}\prod_{n=1}^{N}\frac{1}{2}\left(\mathcal{N}(x_{mn} \,|\, 0, \eta^2) + \mathcal{N}(x_{mn} \,|\, 1, \eta^2\right). \tag{1.3}$$

1.4.4 Computing the ELBO

Here we show how to evaluate the ELBO. Note that the stick-breaking and rounding transformations are compositions of invertible functions, $g_\lambda = h_\lambda \circ f$ with $\Psi = f(z; \theta)$ and $X = h_\lambda(\Psi)$. In both cases, f takes in a matrix of independent standard Gaussians (z) and transforms it with the means and variances in θ to output a matrix Ψ with entries $\psi_{mn} \sim \mathcal{N}(\mu_{mn}, \nu_{mn}^2)$. Stick-breaking

and rounding differ in the temperature-controlled transformations $h_\lambda(\Psi)$ they use to map Ψ toward the Birkhoff polytope.

To evaluate the ELBO, we must compute the density of $q_\lambda(X;\theta)$. Let $J_{h_\lambda}(u) = \frac{\partial h_\lambda(U)}{\partial U}\big|_{U=u}$ denote the Jacobian of a function h_λ evaluated at value u. By the change of variables theorem and properties of the determinant,

$$\begin{aligned} q_\lambda(X;\theta) &= p\big(h_\lambda^{-1}(X);\theta\big) \times \big|J_{h_\lambda^{-1}}(X)\big| \\ &= p\big(h_\lambda^{-1}(X);\theta\big) \times \big|J_{h_\lambda}(h_\lambda^{-1}(X))\big|^{-1}. \end{aligned}$$

Now we appeal to the law of the unconscious statistician to compute the entropy of $q_\lambda(X;\theta)$,

$$\begin{aligned} &\mathbb{E}_{q_\lambda(X;\theta)}\Big[-\log q(X;\theta)\Big] \\ &\quad= \mathbb{E}_{p(\Psi;\theta)}\Big[-\log p(\Psi;\theta) + \log|J_{h_\lambda}(\Psi)|\Big] \\ &\quad= \mathbb{H}(\Psi;\theta) + \mathbb{E}_{p(\Psi;\theta)}\Big[|J_{h_\lambda}(\Psi)|\Big]. \end{aligned} \tag{1.4}$$

Since Ψ consists of independent Gaussians with variances ν_{mn}^2, the entropy is simply,

$$\mathbb{H}(\Psi;\theta) = \frac{1}{2}\sum_{m,n}\log(2\pi e\nu_{mn}^2).$$

We estimate the second term of (1.4) using Monte-Carlo samples. For both transformations, the Jacobian has a simple form.

Jacobian of the Stick-Breaking Transformation

Here h_λ consists of two steps: first we map $\Psi \in \mathbb{R}^{N-1\times N-1}$ to $B \in [0,1]^{N-1\times N-1}$ with a temperature-controlled, elementwise logistic function, then we map B to X in the Birkhoff polytope with the stick-breaking transformation.

As with the standard stick-breaking transformation to the simplex, our transformation to the Birkhoff polytope is feed-forward; i.e. to compute x_{mn} we only need to know the values of β up to and including the (m,n)-th entry. Consequently, the Jacobian of the transformation is triangular, and its determinant is simply the product of its diagonal.

We derive an explicit form in two steps. With a slight abuse of notation, note that the Jacobian of $h_\lambda(\Psi)$ is given by the chain rule,

$$J_{h_\lambda}(\Psi) = \frac{\partial X}{\partial \Psi} = \frac{\partial X}{\partial B}\frac{\partial B}{\partial \Psi}.$$

Since both transformations are bijective, the determinant is,

$$\left|J_{h_\lambda}(\Psi)\right| = \left|\frac{\partial X}{\partial B}\right| \left|\frac{\partial B}{\partial \Psi}\right|.$$

the product of the individual determinants. The first determinant is,

$$\left|\frac{\partial X}{\partial B}\right| = \prod_{m=1}^{N-1}\prod_{n=1}^{N-1}\frac{\partial x_{mn}}{\partial \beta_{mn}} = \prod_{m=1}^{N-1}\prod_{n=1}^{N-1}(u_{mn} - \ell_{mn}).$$

The second transformation, from Ψ to B, is an element-wise, temperature-controlled logistic transformation such that,

$$\begin{aligned}\left|\frac{\partial B}{\partial \Psi}\right| &= \prod_{m=1}^{N-1}\prod_{n=1}^{N-1}\frac{\partial \beta_{mn}}{\partial \psi_{mn}} \\ &= \prod_{m=1}^{N-1}\prod_{n=1}^{N-1}\frac{1}{\lambda}\sigma\left(\psi_{mn}/\lambda\right)\sigma\left(-\psi_{mn}/\lambda\right).\end{aligned}$$

It is important to note that the transformation that maps $B \to X$ is only piecewise continuous: the function is not differentiable at the points where the bounds change; for example, when changing B causes the active upper bound to switch from the row to the column constraint or vice versa. In practice, we find that our stochastic optimization algorithms still perform reasonably in the face of this discontinuity.

Jacobian of the Rounding Transformation

Recall that the rounding transformation is given as follows:

$$x_{mn} = [h_\lambda(\Psi)]_{mn} = \lambda\psi_{mn} + (1-\lambda)[\mathsf{round}(\Psi)]_{mn}.$$

This transformation is piecewise linear with jumps at the boundaries of the "Voronoi cells;" i.e., the points where $\mathsf{round}(X)$ changes. The set of discontinuities has Lebesgue measure zero so the change of variables theorem still applies. Within each Voronoi cell, the rounding operation is constant, and the Jacobian is,

$$\log \left| J_{h_\lambda}(\Psi) \right| = \sum_{m,n} \log \tau = N^2 \log \tau.$$

For the rounding transformation with given temperature, the Jacobian is constant.

1.4.5 Comparison of Proposed Methods

The stick-breaking and rounding transformations introduced above each have their strengths and weaknesses. Here we list some of their conceptual differences. While these considerations aid in understanding the differences between the two transformations, the ultimate test is in their empirical performance, which we describe in Section 1.5.1.

- Stick-breaking relaxes to the Birkhoff polytope whereas rounding relaxes to $\mathbb{R}^{N \times N}$. The Birkhoff polytope is intuitively appealing, but as long as the likelihood, $p(y \,|\, X)$, accepts real-valued matrices, either may suffice.
- Rounding uses the $O(N^3)$ Hungarian algorithm in its sampling process, whereas stick-breaking has $O(N^2)$ complexity. In practice, the stick-breaking computations are slightly more efficient.
- Rounding can easily incorporate constraints. If certain mappings are invalid, i.e. $x_{mn} \equiv 0$, they are given an infinite cost in the Hungarian algorithm. This is hard to do this with stick-breaking as it would change the computation of the upper and lower bounds.
- Stick-breaking introduces a dependence on ordering. While the mapping is bijective, a desired distribution on the Birkhoff polytope may require a complex distribution for B. Rounding, by contrast, is more "symmetric" in this regard.

In summary, stick-breaking offers an intuitive advantage - an exact relaxation to the Birkhoff polytope - but it suffers from its sensitivity to ordering and its inability to easily incorporate constraints. As we show next, in practice these concerns ultimately lead us to favor the rounding based methods.

1.5 Experiments

1.5.1 Synthetic Experiments

We are interested in two principal questions: (i) how well can the stick-breaking and rounding re-parameterizations of the Birkhoff polytope approximate the true posterior distribution over permutations in tractable, low-dimensional cases; and (ii) when do our continuous relaxations offer advantages over alternative Bayesian permutation inference algorithms?

To assess the quality of our approximations for distributions over permutations, we considered a toy matching problem in which we are given the locations of N cluster centers and a corresponding set of N observations, one for each cluster, corrupted by Gaussian noise. Moreover, the observations are permuted so there is no correspondence between the order of observations and the order of the cluster centers. The goal is to recover the posterior distribution over permutations. For $N = 6$, we can explicitly enumerate the $N! = 720$ permutations and compute the posterior exactly.

As a baseline, we consider the Mallows distribution [45] with density over the permutations ϕ given by $p_{\theta,\phi_0}(\phi) \propto \exp(-\theta d(\phi, \phi_0))$, where ϕ_0 is a central permutation with distance between permutations given by $d(\phi, \phi_0) = \sum_{i=1}^{N} |\phi(i) - \phi_0(i)|$, and θ controls the spread around ϕ_0. This is the most popular exponential family model for permutations but, since it is necessarily unimodal, it can fail to capture complex permutation distributions.

Table 1.1: Mean Battacharya distances in the synthetic matching experiment for various methods and observation variances.

	Variance σ^2			
Method	$.1^2$	$.25^2$	$.5^2$	$.75^2$
Stick-breaking	.09	.23	.41	.55
Rounding	**.06**	**.21**	**.32**	**.38**
Mallows ($\theta = 0.1$)	.93	.92	.89	.85
Mallows ($\theta = 0.5$)	.51	.53	.61	.71
Mallows ($\theta = 2$)	.23	.33	.53	.69
Mallows ($\theta = 5$)	.08	.27	.54	.72
Mallows ($\theta = 10$)	.08	.27	.54	.72

We measured the discrepancy between true posterior and an empirical estimate of the inferred posteriors using the Battacharya distance (BD). We fit $q_\lambda(X; \theta)$ with a fixed λ (a hyperparameter) for both stick-breaking and rounding transformations, sampled the variational posterior, and

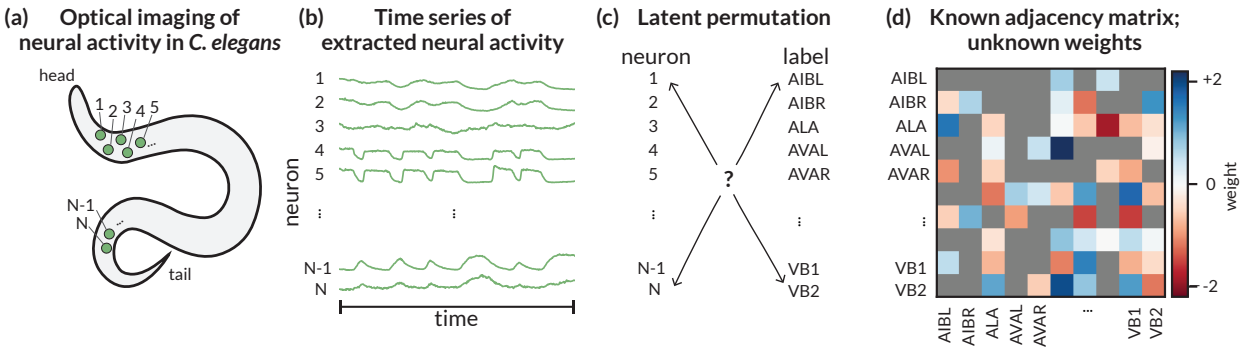


Figure 1.3: Inferring labels and weights in *C. elegans*. (a) Neural activity is optically recorded in genetically modified *C. elegans*. (b) The output is a multivariate time series of neural activity of N neurons for each worm. (c) The first challenge is to infer a latent permutation that matches observed neuron indices to the known set of neuron names, or labels. (d) The second challenge is to infer the weights with which each neuron influences its synaptic neighbors. The connectome (i.e. adjacency matrix) is known, but the weights are not.

rounded the samples to the nearest permutation matrix with the Hungarian algorithm. For the Mallows distribution, we set ϕ_0 to the MAP estimate (also found with the Hungarian algorithm) and sampled using MCMC.

Both methods outperform the simple Mallows distribution and reasonably approximate non-trivial distributions over permutations. Figure 1.2 shows (a) sample experiment configurations; (b) examples of inferred discrete posteriors for stick-breaking, rounding, and Mallows at increasing levels of noise; and (c) histogram of Battacharya distance. The latter are summarized in Table 1.1.

1.5.2 Brain Dynamics of *C. elegans*

We consider an application motivated by the study of the neural dynamics in *C. elegans*. This worm presents many advantages for scientific study. Each hermaphrodite worm has the same $N = 302$ neurons, and each neuron has a label, like AIBL, AVAL, etc. Moreover, the worm's *connectome* - the adjacency matrix that specifies how neurons are connected - is well-characterized [29, 46]. Yet while the adjacency matrix is known, the *weights* associated with these connections are not.

Modern recording technologies enable whole-brain recordings in *C. elegans* [28], presenting an opportunity to learn these weights. Figure 1.3a and Figure 1.3b provide a cartoon illustration: worms are genetically altered so that their neurons fluoresce when active; in each frame of the movie neurons appear as dots in the image, and over time the intensity of these dots provides an optical read-out of the neural activity. However, labeling neurons - i.e. finding the latent

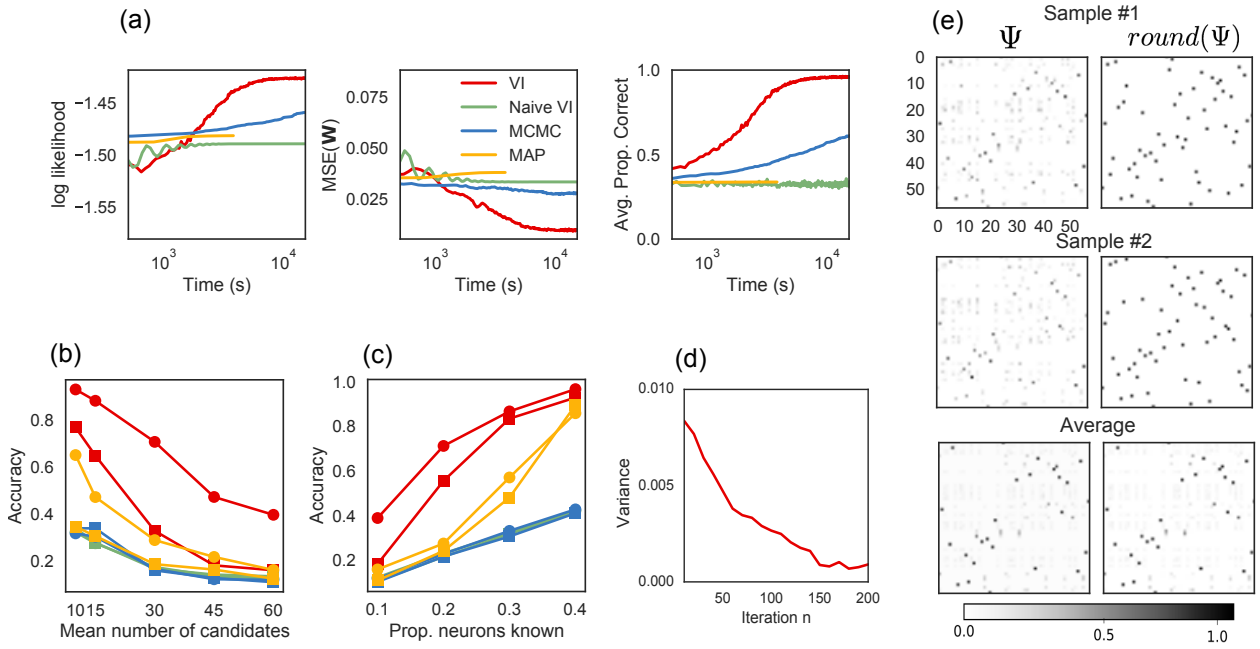


Figure 1.4: Results on the C.elegans inference example. (a) An example of convergence of the algorithm, and the baselines. (b) Accuracy of identity inference as a function of mean number of candidates (correlated with ν), for $M = 1$ worm (square) and combining information of $M = 5$ worms (circles). (c) Accuracy as a function of the proportion of known networks beforehand, with $\nu = 0.1$ (circles) and $\nu = 0.05$ (squares). (d)Variance of distribution over permutations (vectorized) as a function of the number of iterations. (e) Two samples of permutation matrices $\mathsf{round}(\Psi)$ (right) and their noisy, non-rounded versions Ψ (left) at the twentieth algorithm iteration. The average of many samples is also shown. These averages take values in $(0, 1)$, indicating uncertainty in the variational posterior.

permutation that matches observed neurons to known labels, as in Figure 1.3c - is still a manual task. Experimenters consider the location of the neuron along with its pattern of activity to perform this matching, but the process is laborious and the results are prone to error. We prototype an alternative solution, leveraging the location of neurons and their activity in a probabilistic model. We infer neural labels by combining information from the connectome, covariates like position, and neural dynamics. Moreover, we combine information from many individual worms to aid in this labeling. The hierarchical nature of this problem and the plethora of prior constraints and observations motivate our Bayesian approach.

1.5.3 Probabilistic Model

Let J denote the number of worms and $Y^{(j)} \in \mathbb{R}^{T_j \times N}$ denote a recording of worm j with T_j time steps and N neurons. We model the neural activity as a linear autoregressive process

$Y_t^{(j)} = X^{(j)} W X^{(j)\mathsf{T}} Y_{t-1}^{(j)} + \varepsilon_t^{(j)}$, where $\varepsilon_t^{(j)}$ is Gaussian. Here, $X^{(j)}$ is a latent permutation that must be inferred for each worm in order to align the per-worm observations with the shared dynamics matrix W. The hierarchical component of the model is that the unknown weight matrix W is shared by all worms, and it encodes the influence of one neuron on another. The rows and columns of W are ordered in the same way as the known connectome $A \in \{0,1\}^{N \times N}$. The connectome specifies which entries of W may be non-zero: without a connection ($A_{mn} = 0$) the corresponding weight must be zero; if a connection exists ($A_{mn} = 1$), we must infer its weight. A cartoon example is shown in Figure 1.3d.

Our goal is to infer W and $\{X^{(j)}\}$ given $\{Y^{(j)}\}$ and A using variational permutation inference. We place a standard Gaussian prior on W and a uniform prior on $X^{(j)}$, and we use the rounding transformation to approximate the posterior as follows:

$$p(W, \{X^{(j)}\} \mid \{Y^{(j)}\}, A) \propto p(W \mid A) \prod_j p(Y^{(j)} \mid W, X^{(j)}, A)\, p(X^{(j)}).$$

Finally, we use neural position along the worm's body [47] to constrain the possible neural identities for a given neuron. Given the positions of the neurons, we construct a binary *constraint* matrix $C^{(j)}$ so that $C_{mn}^{(j)} = 1$ if observed neuron m is close to where label n is typically found. We enforce this constraint by zeroing corresponding entries in the parameter matrix M described in Section 1.4.2. These constraints greatly reduce the number parameters of the model and facilitate inference.

1.5.4 Results

We compared against three methods: (i) naive variational inference, where we do not enforce the constraint that $X^{(j)}$ be a permutation and instead treat each row of $X^{(j)}$ as a Dirichlet distributed vector; (ii) MCMC, where we alternate between sampling from the conditionals of W (Gaussian) and $X^{(j)}$, from which one can sample by proposing local swaps, as described in [48], and (iii) maximum a posteriori estimation (MAP). Our MAP algorithm alternates between the optimizing estimate of W given $\{X^{(m)}, Y^{(m)}\}$ using linear regression and finding the optimal $X^{(j)}$. The second step requires solving a quadratic assignment problem (QAP) in $X^{(j)}$; that is, it can be expressed as $\mathrm{Tr}(AXBX^{\mathsf{T}})$ for matrices A, B. We used the QAP solver proposed by [49].

We find that our method outperforms each baseline. Figure 1.4a illustrates convergence to a better solution for a certain parameter configuration. Moreover, Figure 1.4b and Figure 1.4c show that our method outperforms alternatives when there are many possible candidates and when only a small proportion of neurons are known with certainty. Figure 1.4c also shows that these Bayesian methods benefit from combining information across many worms.

Altogether, these results indicate our method enables a more efficient use of information than its alternatives. This is consistent with other results showing faster convergence of variational inference over MCMC [7], especially with simple Metropolis-Hastings proposals. We conjecture that MCMC would eventually obtain similar if not better results, but the local proposals—swapping pairs of labels—leads to slow convergence. On the other hand, Figure 1.4a shows that our method converges much more quickly while still capturing a distribution over permutations, as shown by the overall variance of the samples in Figure 1.4d and the individual samples in Figure 1.4e.

For experiments on synthetic matching and the *C. elegans* example we used Autograd [50], explicitly avoiding propagating gradients through the non-differentiable round operation, which requires solving a matching problem.

We used ADAM [51] with learning rate 0.1 for optimization. For rounding, the parameter vector V defined in 3.2 was constrained to lie in the interval $[0.1, 0.5]$. Also, for rounding, we used ten iterations of the Sinkhorn-Knopp algorithm, to obtain points in the Birkhoff polytope. For stick-breaking the variances ν defined in 3.1 were constrained between 10^{-8} and 1. In either case, the temperature, along with maximum values for the noise variances were calibrated using a grid search on the interval $[10^{-2}, 1]$. Improvements may be obtained with the use of an annealing schedule.

In the *C. elegans* example we considered the symmetrized version of the adjacency matrix described in [46]; i.e. we used $A' = (A + A^\top)/2$, and the matrix W was chosen antisymmetric, with entries sampled randomly with the sparsity pattern dictated by A'. To avoid divergence, the matrix W was then re-scaled by 1.1 times its spectral radius. This choice, although not essential, induced a reasonably well-behaved linear dynamical system, rich in non-damped oscillations. We used a time window of $T = 1000$ time samples, and added spherical standard noise at each time. All results in Figure 4 are averages over five experiment simulations with different sampled matrices W. For results in Figure 4b we considered either one or four worms (squares and circles, respectively),

and for the x-axis we used the values $\nu \in \{0.0075, 0.01, 0.02, 0.04, 0.05\}$. We fixed the number of known neuron identities to 25 (randomly chosen). For results in Figure 4c we used four worms and considered two values for ν; 0.1 (squares) and 0.05 (circles). Different x-axis values correspond to fixing 110, 83, 55 and 25 neuron identities.

1.6 Discussion

Our results provide evidence that variational permutation inference is a valuable tool, especially in complex problems like neural identity inference where information must be aggregated from disparate sources in a hierarchical model. As we apply this to real neural recordings, we must consider more realistic, nonlinear models of neural dynamics. Here, again, we expect variational methods to shine, leveraging automatic gradients of the relaxed ELBO to efficiently explore the space of variational posterior distributions.

1.7 Corresponding Methods of Discrete Variational Inference for Categorical Distributions

We can gain insight and intuition about the stick-breaking and rounding transformations by considering their counterparts for discrete, or categorical, variational inference.

1.7.1 Stick-Breaking

The stick-breaking transformation to the Birkhoff polytope presented in Section 1.4 contains a recipe for stick-breaking on the simplex. In particular, as we filled in the first row of the doubly-stochastic matrix, we were transforming a real-valued vector $\psi \in \mathbb{R}^{N-1}$ to a point in the simplex. We present this procedure for discrete variational inference again here in simplified form. Start with a reparameterization of a Gaussian vector,

$$\begin{aligned} z_n &\overset{\text{iid}}{\sim} \mathcal{N}(0,1), \\ \psi_n &= \mu_n + \nu_n z_n, \qquad 1 \le n \le N-1, \end{aligned}$$

parameterized by $\theta = (\mu_n, \nu_n)_{n=1}^{N-1}$. Then map this to the unit hypercube in a temperature-controlled manner with the logistic function,

$$\beta_n = \sigma(\psi_n/\lambda),$$

where $\sigma(u) = (1 + e^{-u})^{-1}$ is the logistic function. Finally, transform the unit hypercube to a point in the simplex:

$$\begin{aligned} x_1 &= \beta_1, \\ x_n &= \beta_n \left(1 - \sum_{m=1}^{n-1} x_m\right), \qquad 2 \le n \le N-1, \\ x_N &= 1 - \sum_{m=1}^{N-1} x_m, \end{aligned}$$

Here, β_n is the fraction of the remaining "stick" of probability mass assigned to x_n. This transformation is invertible, the Jacobian is lower-triangular, and the determinant of the Jacobian is easy to compute. In [52], the density of x implied by a Gaussian density on ψ is computed.

The temperature λ controls how concentrated $p(x)$ is at the vertices of the simplex, and with appropriate choices of parameters, in the limit $\lambda \to 0$ we can recover any categorical distribution (we will discuss this in detail in Section 1.7.3). In the other limit, as $\lambda \to \infty$, the density concentrates on a point in the interior of the simplex determined by the parameters, and for intermediate values, the density is continuous on the simplex.

Finally, note that the logistic-normal construction is only one possible choice. We could instead let $\beta_n \sim \text{Beta}(\frac{a_n}{\lambda}, \frac{b_n}{\lambda})$. This would lead to a generalized Dirichlet distribution on the simplex. The beta distribution is slightly harder to reparameterize since it is typically simulated with a rejection sampling procedure, but [53] have shown how this can be handled with a mix of reparameterization and score-function gradients. Alternatively, the beta distribution could be replaced with the Kumaraswamy distribution [54], which is quite similar to the beta distribution but is easily reparameterizable.

1.7.2 Rounding

Rounding transformations also have a natural analog for discrete variational inference. Let e_n denote a one-hot vector with n-th entry equal to one. Define the rounding operator,

$$\mathsf{round}(\psi) = e_{n^*},$$

where

$$\begin{aligned} n^* &= \arg\min_n \|e_n - \psi\|^2 \\ &= \arg\max_n \psi_n. \end{aligned}$$

In the case of a tie, let n^* be the smallest index n such that $\psi_n > \psi_m$ for all $m < n$. Rounding effectively partitions the space into N disjoint "Voronoi" cells,

$$V_n = \Big\{\psi \in \mathbb{R}^N : \psi_n \geq \psi_m \, \forall m \ \wedge \ \psi_n > \psi_m \, \forall m < n\Big\}.$$

By definition, $\mathsf{round}(\psi) = e_{n^*}$ for all $\psi \in V_{n^*}$

We define a map that pulls points toward their rounded values,

$$x = \lambda\psi + (1-\lambda)\mathsf{round}(\psi). \tag{1.5}$$

Proposition 1.1. *For $\lambda \in [0,1]$, the map defined by (1.5) moves points strictly closer to their rounded values so that $\mathsf{round}(\psi) = \mathsf{round}(x)$.*

Proof. Note that the Voronoi cells are intersections of halfspaces and, as such, are convex sets. Since x is a convex combination of ψ and e_{n^*}, both of which belong to the convex set V_{n^*}, x must belong to V_{n^*} as well. □

Similarly, x will be a point on the simplex if and only if ψ is on the simplex as well. By analogy to the rounding transformations for permutation inference, in categorical inference we use a Gaussian distribution $\psi \sim \mathcal{N}(\mathsf{proj}(m), \nu)$, where $\mathsf{proj}(m)$ is the projection of $m \in \mathbb{R}_+^N$ onto the simplex. Still, the simplex has zero measure under the Gaussian distribution. It follows that the

rounded points x will almost surely not be on the simplex either. The supposition of this approach is that this is not a problem: relaxing to the simplex is nice but not required.

In the zero-temperature limit we obtain a discrete distribution on the vertices of the simplex. For $\lambda \in (0, 1]$ we have a distribution on $\mathcal{X}_\lambda \subseteq \mathbb{R}^N$, the subset of the reals to which the rounding operation maps. (For $0 \leq \lambda < 1$ this is a strict subset of $\mathbb{R}^N$.) To derive the density $q(x)$, we need the inverse transformation and the determinant of its Jacobian. From Proposition 1.1, it follows that the inverse transformation is given by,

$$\psi = \frac{1}{\lambda} x - \frac{1-\lambda}{\lambda} \mathsf{round}(x).$$

As long as ψ is in the interior of its Voronoi cell, the round function is piecewise constant and the Jacobian is $\frac{\partial \psi}{\partial x} = \frac{1}{\lambda} I$, and its determinant is λ^{-N}. Taken together, we have,

$$q(x; m, \nu) = \lambda^{-N} \mathcal{N}\left(\frac{1}{\lambda} x - \frac{1-\lambda}{\lambda} \mathsf{round}(x);\, \mathsf{proj}(m), \mathbf{f}diag(\nu)\right) \times \mathbb{I}[x \in \mathcal{X}_\lambda].$$

1.7.3 Limiting Analysis for Stick-Breaking

We show that stick-breaking for discrete variational inference can converge to any categorical distribution in the zero-temperature limit.

Let $\beta = \sigma(\psi/\lambda)$ with $\psi \sim \mathcal{N}(\mu, \nu^2)$. In the limit $\lambda \to 0$ we have $\beta \sim \text{Bern}(\Phi(-\frac{\mu}{\nu}))$, where $\Phi(\cdot)$ denotes the Gaussian cumulative distribution function (cdf). Moreover, when $\beta_n \sim \text{Bern}(\rho_n)$ with $\rho_n \in [0, 1]$ for $n = 1, \ldots, N$, the random variable x obtained from applying the stick-breaking transformation to β will have an atomic distribution with atoms in the vertices of Δ_N; i.e, $x \sim \text{Cat}(\pi)$ where

$$\begin{aligned}
\pi_1 &= \rho_1 \\
\pi_n &= \rho_n \prod_{m=1}^{n-1} (1 - \rho_m) \qquad n = 2, \ldots, N-1, \\
\pi_N &= \prod_{m=1}^{N-1} (1 - \rho_m).
\end{aligned}$$

These two facts, combined with the invertibility of the stick-breaking procedure, lead to the following proposition

Proposition 1.2. *In the zero-temperature limit, stick-breaking of logistic-normal random variables can realize any categorical distribution on x.*

Proof. There is a one-to-one correspondence between $\pi \in \Delta_N$ and $\rho \in [0,1]^{N-1}$. Specifically,

$$\rho_1 = \pi_1$$
$$\rho_n = \frac{\pi_n}{\prod_{m=1}^{n=1} 1 - \rho_m} \quad \text{for } n = 2, \ldots, N-1.$$

Since these are recursively defined, we can substitute the definition of ρ_m to obtain an expression for ρ_n in terms of π only. Thus, any desired categorical distribution π implies a set of Bernoulli parameters ρ. In the zero temperature limit, any desired ρ_n can be obtained with appropriate choice of Gaussian mean μ_n and variance ν_n^2. Together these imply that stick-breaking can realize any categorical distribution when $\lambda \to 0$. □

1.7.4 Variational Autoencoders (VAE) with Categorical Latent Variables

We considered the density estimation task on MNIST digits, as in [12, 13], where observed digits are reconstructed from a latent discrete code. We used the continuous ELBO for training, and evaluated performance based on the marginal likelihood, estimated with the variational objective of the discretized model. We compared against the methods of [12, 13] and obtained the results in Table 1.2. We used Tensorflow [55] for the these experiments, slightly changing the code made available from [13].

Table 1.2: Summary of results in VAE

Method	$-\log p(x)$
Concrete	111.5
Rounding	121.1
Stick-breaking	119. 8

Figure 1.5 shows MNIST reconstructions using Concrete, stick-breaking and rounding reparameterizations. In all the three cases reconstructions are reasonably accurate, and there is diversity in reconstructions.

While stick-breaking and rounding fare slightly worse than the Concrete distribution, they are readily extensible to more complex discrete objects, as previously shown. However, this difference

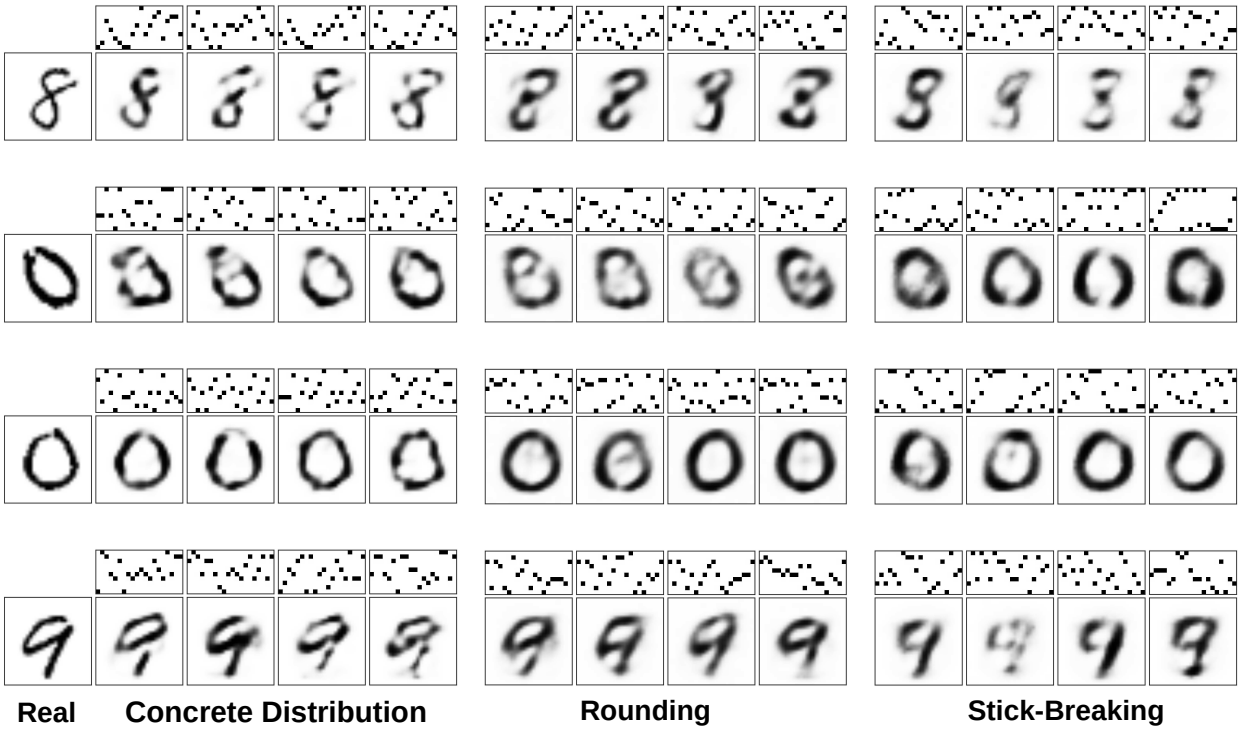


Figure 1.5: Examples of true and reconstructed digits from their corresponding discrete latent variables. The real input image is shown on the left, and we show sets of four samples from the posterior predictive distribution for each discrete variational method: Concrete, rounding, and stick-breaking. Above each sample we show the corresponding sample of the discrete latent "code." The random codes consist of of $K = 20$ categorical variables with $N = 10$ possible values each. The codes are shown as 10×20 binary matrices above each image.

in performance and increase in complexity is justification for primarily focusing on the Concrete distribution in Chapter 3 and Chapter 5 when focusing on Categorical distributions as part of the development of the SmartGraph system. In Chapter 5, we develop an approximation to the Concrete distribution that makes it computationally tractable for use in a graph setting.

The "network" aspects of this current chapter are quite clear, as are the relevance of deep learning techniques to solving problems related to networks. However, as we observed, there is a lot of manual work required in setting up approximate priors and finding appropriate transformations for each problem, that may be prohibitive in any context where we may wish to know many aspects of a system, and do not have the capability to perform such specific analytics for each and every problem variation. An example of such a situation is a graph database, with many users wishing to run different graph analytics and computation using shared and/or distinct data. Nonetheless, the techniques of this chapter form the basis for how machine learning problems can be approached

on graphs in a fairly general way through the use of continuous approximation techniques, as demonstrated in future chapters.

Chapter 2

Learning Graph Topological Features through Generative Adversarial Networks

In this chapter, we explore some typical approaches for handling graph structured datasets for machine learning tasks. Key focus points of this chapter are how graphs are interpreted by traditional deep learning methodologies, and how the disparity in the structure of graphs necessitates more flexible approaches that can handle many types of graphs. This chapter focuses on structural considerations more than vertex and edge properties, and demonstrates a methodology capable of dealing with many graphical structures. This work was key in our understanding the need for a machine learning methodology for graphs that can not only be used for a wide variety of graphs, but can also interpret graphs in a more natural way (e.g. through querying), a concept developed further in Chapters 3-5.

2.1 Motivation

Inspired by the power of generative adversarial networks (GANs) in image domains, we introduce [2, 56] a novel hierarchical architecture for learning characteristic topological features from a single arbitrary input graph via GANs. The hierarchical architecture consisting of multiple GANs preserves both local and global topological features and automatically partitions the input graph

into representative "stages" for feature learning. The stages facilitate reconstruction and can be used as indicators of the importance of the associated topological structures. Experiments show that our method produces subgraphs retaining a wide range of topological features, even in early reconstruction stages (unlike a single GAN, that cannot easily identify such features, let alone reconstruct the original graph). This chapter outlines the first research effort on combining the use of GANs and graph topological analysis.

2.2 Introduction

Graphs have great versatility, able to represent complex systems with diverse relationships between objects and data. With the rise of social networking, and the importance of relational properties to the "big data" phenomenon, it has become increasingly important to develop ways to automatically identify key structures present in graph data. Identification of such structures is crucial in understanding how a social network forms, or in making predictions about future network behavior. To this end, numerous graph analysis methods have been proposed to analyze the topology of the target network at the node [57], community [58, 59], and global levels [60], and perform certain data analysis and machine learning tasks.

Unfortunately, each level of analysis is greatly influenced by the underlying network topology, and so far no algorithm can be effectively and automatically adapted to arbitrary and complex network structures. For example, modularity-based community detection [61] works well for networks with separate clusters, whereas edge-based methods [62] prevail in dense networks. Similarly, when performing graph sampling, Random Walk (RW) is suitable for sampling paths [63], whereas Forrest Fire (FF) is useful for sampling clusters [64]. When it comes to graph generation, Watts-Strogatz (WS) graph models [65] can generate graphs with small world features, whereas Barabasi-Albert (BA) graph models [66] can simulate super hubs and regular nodes according to the scale-free features of the network.

However, real-world networks typically have multiple and potentially complex topological features, which may be beyond the expressive power of a single graph model. Moreover, taking real-world networks into consideration also introduces another issue that traditional graph analysis methods have been struggling with: one may only have a single instance of a graph (e.g. the transaction graph for a particular bank), making it difficult to identify the key topological properties

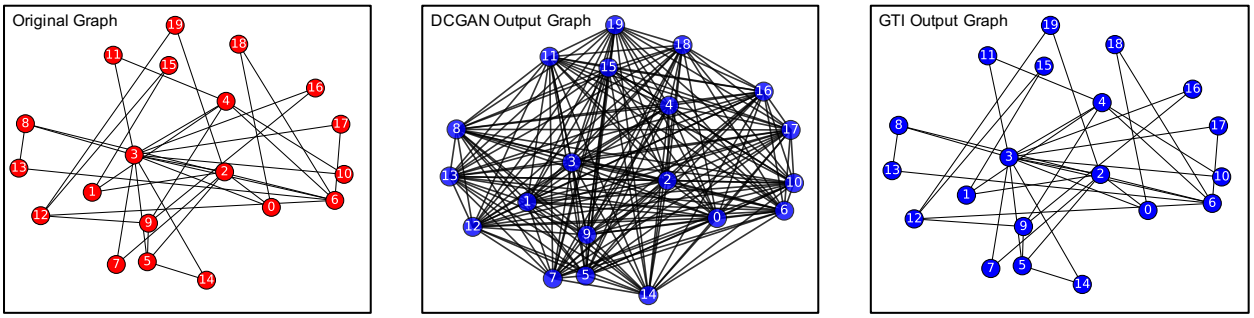


Figure 2.1: How GTI recovers the original graph while naive GAN methods fail.

in the first place. In particular, many graph mining and analysis problems are often associated with both "local topological features" such as the presence of subgraph structures like triangles and "global topological features" such as degree distribution.

In this chapter, we propose an unsupervised method, the Graph Topology Interpolater (GTI), to facilitate graph analysis with an aim of bypassing the aforementioned issues. GTI is a novel approach combining techniques from graph analysis and GAN based image processing techniques. In Figure 2.1, we demonstrate that naively feeding the full graph (here, a 20 node BA network [66]) into a standard GAN implementation (the DCGAN [67]) is unsuccessful; such a GAN structure is unable to learn to reproduce the original graph, and instead becomes stuck in undesirable local minima. By reproducing critical topological structures through stages, we are able to *interpretably* demonstrate the importance of topological properties and features to the graph.

Instead of directly and naively analyzing the entire topology of a graph as an image, GTI first divides the graph into several hierarchical layers. A hierarchical view of a graph can split the graph by local and global topological features, giving a better understanding of the graph [68]. As different layers have different topological features, GTI uses separate GANs to learn each layer and the associated features. By leveraging GAN's feature identification capability [69, 70] on each layer, GTI has the ability to automatically capture arbitrary topological features from a single input graph.

In addition to learning topological features from the input graph, the GTI method defines a reconstruction process for reproducing the original graph via a series of reconstruction stages (the number of which is automatically learned during training). As stages are ranked in the order of their contribution to the full topology of the original graph, early stages can be used as indicators of the most important topological features. This work is the first-line research in leveraging GAN

to learn hierarchical topological features given a single input graph. We demonstrate its ability to efficiently learn important topological features to retain critical structures and make comparisons to graph sampling methods.

Our proposed GTI method has three core contributions:

1. GTI is a model-agnostic graph topology analysis tool, which means it can analyze arbitrary kinds of topology for a graph, rank the edge set, and outputs representative stages for the graph.
2. For each stage, it can preserve both local and global topological features of the input graph.
3. The input of GTI is a single graph, which renders this method applicable to a wide range of applications with unique but rare graphs.

2.3 Method

In this section, we demonstrate the work flow of our Graph Topology Interpolater (GTI) (see Figure 2.2), with a particular focus on the GAN, Sum-up and Stage Identification modules. At a high level, the GTI method takes an input graph, learns its hierarchical layers, trains a separate GAN on each layer, and autonomously combines their output to reconstruct stages of the graph. Here we give a brief overview of each module.

Hierarchical Identification Module: This module detects the hierarchical structure of the original graph using the Louvain hierarchical community detection method [71]. Let L denote the number of layers. The average size of communities in each layer is used as a criterion for how many subgraphs a layer should pass to the next module.

Layer Partition Module: The main purpose of this module is to partition a given layer into M non-overlapping subgraphs, where M is the number of subgraphs. The reason why we do not use the learned communities from the Louvain method is that we cannot constrain the size of any community. We instead balance the communities into fixed size subgraphs using the METIS approach [72].

Layer GAN Module: Here we apply the GAN methodology to graph analysis. Rather than directly using one GAN to learn the whole graph, we use different GANs to learn features for each

layer separately. If we use a single GAN to learn features for the whole graph, some topological features may be diluted or even ignored. See Section 2.3.2 for more details.

Layer Regeneration Module: Here, for a given layer, the corresponding GAN has learned all the properties of each subgraph, meaning we can use the generator in this GAN to regenerate the topology of the layer by generating M subgraphs consisting of k nodes. Note that this reconstruction only restores edges within each non-overlapping subgraph, and does not include edges between subgraphs.

All-layer Sum-up Module: By summing up the results from reconstructed layers, along with the edges that were not considered in the Regeneration Module, the purpose of this module is to output a weighted graph, where the "weight" represents the importance of an edge during the reconstructing process. Indeed, we rely upon these weights to identify the reconstruction stages for the original graph. For more details, see Section 2.3.3.

Stage Identification Module: By analyzing the weighted adjacency matrix of the Sum-up Module, we can regenerate the graph by controlling different thresholds to generate an edge on a node-pair with respect to the value of the previously obtained weighted adjacency matrix. Here, we use the term "stage" to indicate a regenerated graph where the weights of edges are equal or larger than a given threshold. Note that these stages can be interpreted as steps for graph reconstruction, and can be automatically determined by the weights of edges. See Section 2.3.4 for details.

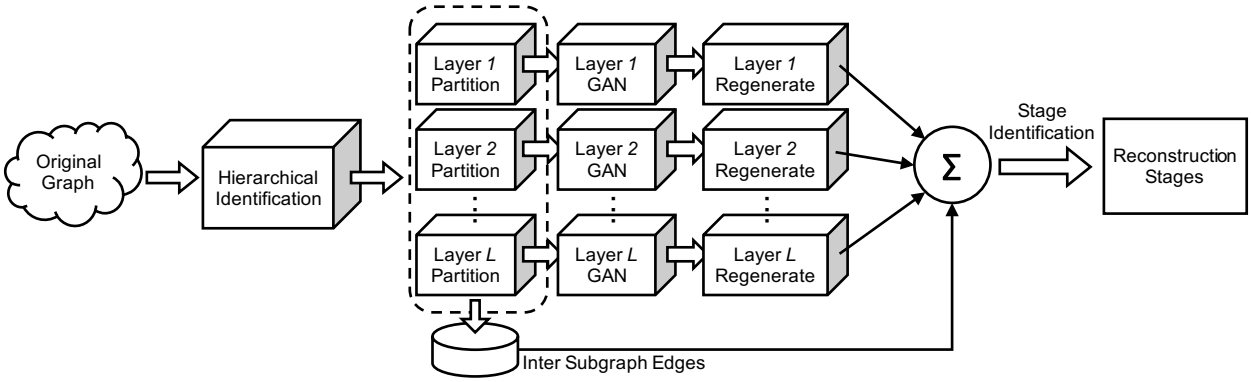


Figure 2.2: Work flow for the Graph Topology Interpolater (GTI).

2.3.1 Layer Identification and Layer Partition Module

This module implements the Louvain community detection method to identify the number of hierarchical layers in a graph [71]. By introducing a local optimization function "Incremental Modularity ΔQ," this approach calculates the ΔQ of a node and its neighbors, and selects one with the largest ΔQ. When $\Delta Q > 0$, the corresponding neighbor(s) are merged to the local community where the current node belongs to. The selection process will stop when the local community in the graph is no longer changing. Following this, the Louvain method begins a new iteration by merging these local communities into a new layer in the hierarchy, and again performing $\arg\max \Delta Q$ to detect new local communities in this layer. Consequently, this algorithm automatically constructs hierarchical layers for a graph.

As the Louvain method helps to determine hierarchical structure and communities of the input graph in an unsupervised manner, we use two sources of information, the number of hierarchies and the number of communities in the corresponding layer, as prior information for guiding the algorithm to generate training samples for each layer. The reason to use the number of communities instead of the communities themselves is because the input tensor for convolution neural network always requires the same size inputs. Hence, we need to unify the size of input samples for the generator to learn. In practice, this harsh requirement can be satisfied by adding isolated node(s) to the subgraph, or padding zeros to the adjacent matrix of the subgraph. Of course, this has the consequence of introducing noise. This sort of strict requirement about input dimensions (for a class of input objects where this varies drastically) is an example of why future chapters of this dissertation involve developing alternative input methods for varied graph structures, for use in the SmartGraph system.

If we directly used the identified community as a subgraph, given the fact that one cannot ensure Louvain method will detect communities of the same size, we would need to add more isolated nodes to make same-sized communities and will inevitably incur the aforementioned noise in the learning process. To avoid this issue, we use the METIS method [72]. The METIS method takes a graph as its input, and partitions the nodes in a balanced way so that each partition has the same number of nodes while minimizing the number of cut edges. Consequently, in the task of community detection with a known number of communities, the METIS method finds same-sized communities with strong within-community connections.

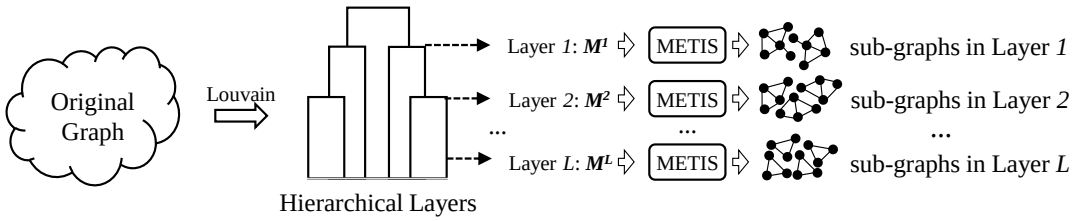


Figure 2.3: An example of Layer Identification and Layer Partition module.

The overall procedure for the Layer Identification and Layer Partition module is summarized in Figure 2.3. First, by conducting Louvain method on the input graph, we can obtain the hierarchical structure. Here, each hierarchy stands for an layer, and we can also obtain the number of communities in this layer (i.e. $M^1, M^2 \cdots M^l$ for layer 1, 2, and l, respectively). Then, for each layer, we use the METIS method, along with the corresponding number of communities, to partition the original graph into same-sized communities (sub-graphs).

2.3.2 Layer GAN Module

Figure 2.4 and Figure 2.5 show the architectures for the generator and discriminator of the GAN, respectively. The generator is a deconvolutional neural network with the purpose of restoring a $k \times k$ adjacency matrix from the standard normal distribution. The discriminator is instead a CNN whose purpose is to estimate if the input adjacency matrix is from a real dataset or from a generator. Here, BN is short for batch normalization which is used instead of max pooling because max pooling selects the maximum value in the feature map and ignores other values, whereas BN will synthesize all available information. LR stands for the leaky ReLU activation function ($LR = max(x, 0.2 \times x)$). Its value 0 carries a specific meaning for adjacency matrices (no edges). In addition, k represents the size of a subgraph, and FC is the length of a fully connected layer. We set the stride for the convolutional/deconvolutional layers to be 2. We adopt the same loss function and optimization strategy as used in DCGAN [67].

2.3.3 Sum-up Module

In this module, we use a linear function (see Equation (2.1)) to add the graphs from all layers together. The term re_G stands for the reconstructed adjacency matrix (with inputs from all layers),

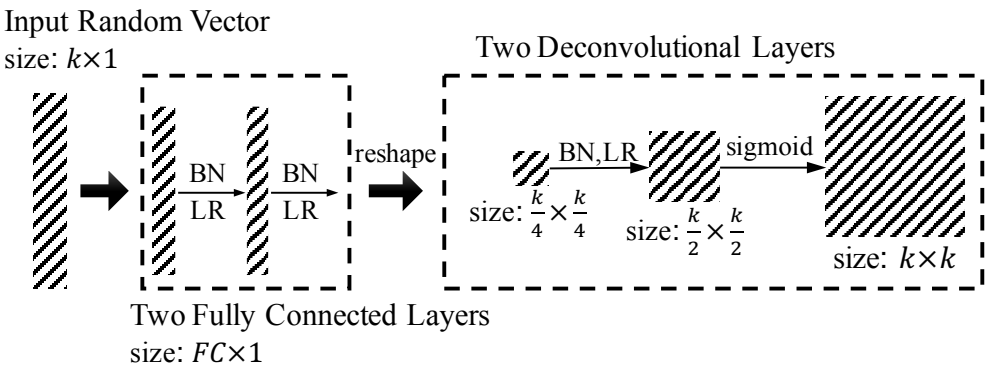


Figure 2.4: Generator architecture.

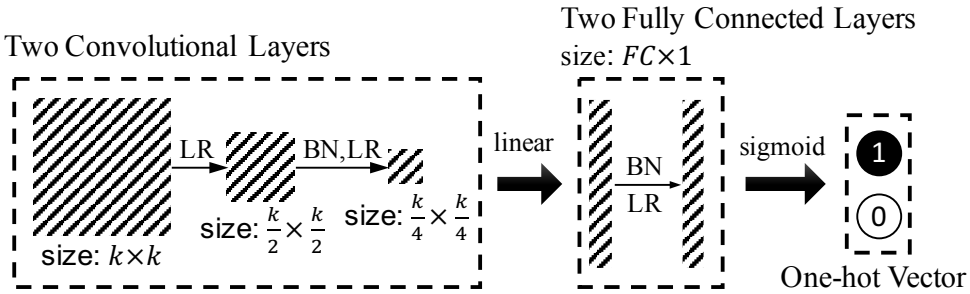


Figure 2.5: Discriminator architecture.

G'_i $(i \in L)$ represents the reconstructed adjacency matrix for each layer (with G representing the full original graph with N nodes), E refers to all the inter-subgraph (community) edges identified by the Louvain method from each hierarchy, and b represents a bias. Note that while each layer of the reconstruction may lose certain edge information, summing up the hierarchical layers along with E will have the ability to reconstruct the entire graph:

$$re_G = \sum_{i=1}^{L} w_i G'_i + wE + b \tag{2.1}$$

To obtain the weight w for each layer and the bias b, we use Equation (2.2) as the loss function (where we add $\epsilon = 10^{-6}$ to avoid taking $log(0)$ or division by 0), use 500 iterations of stochastic gradient descent (SGD) with learning rate 0.1 to minimize this loss function and find suitable parameters. We note that Equation (2.2) is similar to a KL divergence, though of course re_G and

G are not probability distributions.

$$\text{Loss } (re_G, G) = \sum_{i \in 1 \cdots N^2} vec(G + \epsilon)_i \cdot log \frac{vec(G + \epsilon)_i}{vec(re_G + \epsilon)_i} \tag{2.2}$$

2.3.4 Stage Identification

After the above calculations, we obtain the weighted adjacency matrix re_G, where weights on edges represent how each edge contributes to the entire topology. Clearly, different weights represent different degrees of contribution to the topology. Therefore, according to these weights, we can divide the network into several stages, with each stage representing a collection of edges greater than a certain weight. We introduce the concept of a "cut-value" to turn re_G into a binary adjacency matrix. We observe that many edges in re_G share the same weight, which implies these edges share the same importance. Furthermore, the number of unique weights can define different reconstruction stages, with the most important set of edges sharing the highest weight. Each stage will include edges with weights greater than or equal to the corresponding weight of that stage. Hence, we define an ordering of stages by decreasing weight, giving insight on how to reconstruct the original graph in terms of edge importance. We denote the i-th largest unique weight-value as CV_i (for "cut value") and thereby denote the stages as in Equation (2.3) (an element-wise product), where $I[w \geq CV_i]$ is an indicator function for each weight being equal or larger than the CV_i:

$$re_G^i = re_G I[w \geq CV_i] \tag{2.3}$$

In Section 2.5, we use synthetic and real networks to show that each stage preserves identifiable topological features of the original graph during the graph reconstruction process. As each stage contains a subset of the original graphs edges, we can interpret each stage as a sub-sampling of the original graph. This allows us to compare with prominent graph sampling methodologies to emphasize our ability to retain important topological features.

By comparing GTI with some state-of-the-art graph sampling methodologies, we have found that the stages in GTI have similar performance for most metrics. Moreover, we note that since the number of stages in the reconstruction and the size of the graph in each stage are learned automatically, we do not have precise control over the size of the "sampled" graphs. Of course, we

should recall that the purpose of the approach is not sampling as such, but rather to understand the important topological properties of the graph interpretably, as demonstrated by the reconstruction stages.

2.4 Related Work

The development of deep learning and the growing maturity of graph topology analysis has led to more attention on the ability to use the former for the latter [73]. A number of supervised and semi-supervised learning methods have been developed for graph analysis. A particular focus is on the use of CNNs [74–77]. These methods have shown promising results on their respective tasks in comparison to traditional graph analysis methods (such as kernel-based methods, graph-based regularization techniques, etc). The above methods have been discussed in detail, with [77, 78] pointing out the strengths and drawbacks of various approaches. Recently, it was proposed [79] that one could use a naive method to generate the graph topology using GANs, by randomly perturbing the input graph 10,000 times, and feeding these graphs into a DCGAN. It simply treats the adjacency matrix of a graph as an image, and uses random permutation to generate enough samples to train a DCGAN. Unlike our approach, this does not take any topological information of the graph into consideration.

A key difference between GTI and other methods is that GTI is an unsupervised learning tool (facilitated by the use of GANs), that leverages the hierarchical structure of a graph. GTI can automatically capture both local and global topological features of a network. To the best of the authors' knowledge, this is the first unsupervised method in such manner.

Since GANs were first introduced [69] in 2014, its theory and application have expanded greatly. Many advances in training methods [80–83] have been proposed in recent years, and this has facilitated their use in a number of applications. For example, GANs have been used for artwork synthesis [84], text classification [85], image-to-image translation [86], imitation of driver behavior [87], identification of cancers [88], etc. The GTI method expands the use of GANs into the graph topology analysis area.

2.5 Evaluation

All experiments in this chapter were conducted locally on CPU using an Intel Core i7 2.5GHz processor and 16GB of 1600MHz RAM. Though this limits the size of our experiments in this work, the extensive GAN literature (see Section 2.4) and the ability to parallelize GAN training based on hierarchical layers suggests that our method can be efficiently scaled to much larger systems. To study the benefit of the proposed GTI method, this section provides both qualitative and quantitative evaluations of the stages identified by GTI.

2.5.1 Datasets

We use a combination of synthetic and real datasets. Through the use of synthetic datasets with particular topological properties, we are able to demonstrate the ability of retaining these easily identifiable properties across the reconstruction stages using GTI. In addition to validation on synthetic datasets with known topological properties, we also demonstrate our method on a number of real-world datasets with varying sizes.

We use the ER graph model [89], the BA graph model [66], the WS graph model [68] and the Kronecker graph model [90] to generate our synthetic graphs. Here, we have observed that for a WS network, the hyper-parameter k will affect the clustering coefficient, as k controls the number of neighbors a node may connect in a WS network (WS-1). That is, if we set $k \leq 3$, the clustering coefficient for each node in WS network is 0. Hence, we add an extra WS network (WS-2) with $k = 6$ to avoid this case. For all synthetic networks, we use the python package NetworkX [91] to generate the corresponding graph.

For real datasets, we use data available from the Stanford Network Analysis Project (SNAP) [92]. In particular, we use the Facebook network, the Wiki-Vote network, and the P2P-Gnutella network. The Facebook [93] dataset consists of "friends lists", collected from survey participants according to the connections between user-accounts on the online social network. It includes node features, circles, and ego networks; all of which has been anonymized by replacing the Facebook-internal IDs. Wiki-Vote [94] is a voting network (who votes for whom etc) that is used by Wikipedia to elect page administrators; P2P-Gnutella [95] is a peer-to-peer file-sharing network: Nodes represent hosts in the Gnutella network topology, with edges representing connections between the

Table 2.1: Size of original datasets, hyper-parameters for synthetic graphs, and corresponding reconstruction stages

Graph	# Nodes	# Edges	# Stages	Retained edge percentage for ordered stages (%)
BA (m=2)	500	996	7	19.48, 26.31, 36.04, 39.36, 41.57, 57.43, 100
ER (p=0.2)	500	25103	4	4.32, 21.73, 94.91, 100
WS-1 (p=0.2, k=3)	500	500	7	11.20, 11.40, 16.00, 18.00, 54.60, 97.80, 100
WS-2 (p=0.2, k=6)	500	1500	8	15.60, 16.30, 21.04, 25.18, 38.62, 73.21, 98.13, 100
Kronecker	2178	25103	10	87.77, 88.65, 91.76, 91.89, 92.47, 93.32, 96.06, 97.05, 98.57, 100
Facebook	4039	88234	7	52.28, 83.33, 87.49, 91.41, 90.31, 91.95, 100
Wiki-Vote	7115	103689	4	58.31, 73.79, 85.60, 100
RoadNet	5371	7590	12	0.62, 3.87, 26.64, 27.98, 31.79, 32.42, 34.22, 34.65, 34.80, 64.06, 76.81, 100
P2P	3334	6627	7	49.04, 53.90, 70.32, 87.54, 88.40, 89.65, 100

hosts. RoadNet [96] is the road network of Pennsylvania. Intersections and endpoints are represented by nodes, and the roads connecting them are edges. As this graph is of a size prohibitive to the computing resources used in work, we choose a connected component of appropriate size (note that the full network is not connected because nodes may be connected in the real-world by roads outside of Pennsylvania). Detailed information for the synthetic graphs as well as the real-world datasets are outlined in Table 2.1.

2.5.2 Local Topological Features

As discussed in Section 2.3, GTI automatically ranks edge sets based on their contribution to the reconstruction of the original graph. Here we use two examples to demonstrate how this process retains important local topological structure during each reconstruction stage. By applying GTI to the BA network, the method learns that there are six stages for reconstruction of the original topology (with stages 1, 3, and 5 shown in Figure 2.6). In our second example, we demonstrate GTI reconstruction for the real-world RoadNet dataset. For this input, Figure 2.7 demonstrates the learned reconstruction stages 4,8, and 11.

BA Network Stage Analysis: We demonstrate the reconstruction process of a BA network in Figure 2.6, with the top row demonstrating the entire reconstruction process of the full network. We clearly observe that each reconstructed network becomes denser and denser as additional stages are added. The bottom row of Figure 2.6 shows the subgraphs corresponding to nodes 0 to 19 at each reconstruction stage. We observe that these subgraphs retain the most important feature of the original subgraph (the star structures at node 0), even during the first reconstruction stage. In addition, we observe that the final stage exactly corresponds to the original stage. We again note

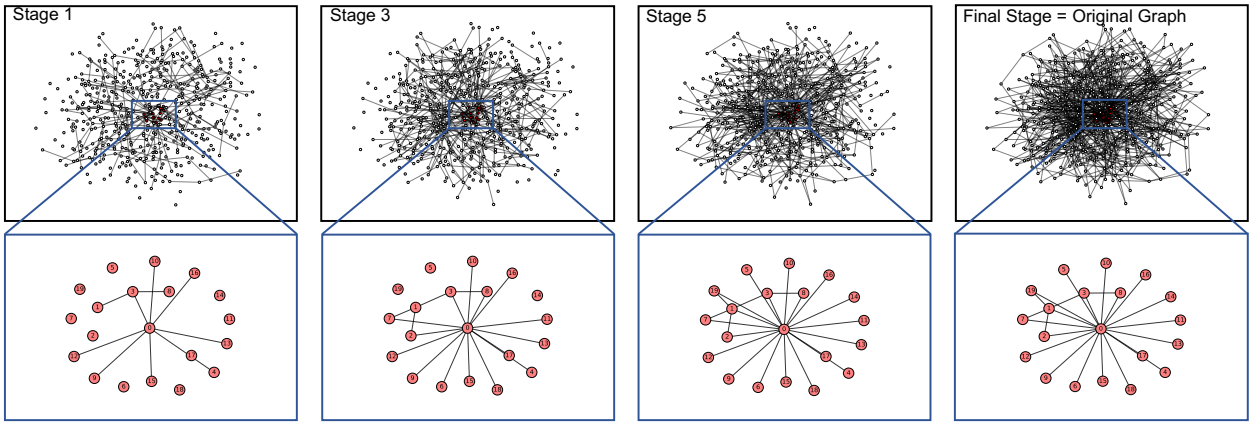


Figure 2.6: The topology of original graph and corresponding stages of BA network.

that as illustrated in Figure 2.1, this reconstruction is extremely difficult when training a single GAN directly on the original graph.

Road Network Stages Analysis: We observe in Table 2.1 that the retained edge percentages of the RoadNet reconstruction decrease more consistently with each stage than in the BA network. This is reasonable, because geographical distance constraints naturally result in fewer super hubs, with each node having less variability in its degree. In Figure 2.7, we observe the reconstruction of the full network, and the node 0 to node 19 subgraph of RoadNet. We observe in the bottom row of Figure 2.7 that the dominant cycle structure of the original node 0-19 subgraph clearly emerges.

We also observe an interesting property of the stages of the original graph in the top row of Figure 2.7. As SNAP does not provide the latitude and longitude of nodes, we cannot use physical locations. We instead calculate the modularity of each stage, where modularity represents how well connected the community is [97]. The tighter the community, the larger the modularity. We found that the modularity deceases from 0.98 to 0.92 approximately linearly. This suggests that the GTI stages first prioritize the clustering of nodes (through edge connections) over connections between clusters. This indicates that GTI views the dense connections between local neighborhoods as a particularly important topological property of road networks, consistent with our intuition.

2.5.3 Global Topological Features

In this section, we demonstrate the ability of GTI reconstruction stages to preserve global topological features, focusing on degree distribution and the distribution of cluster coefficients.

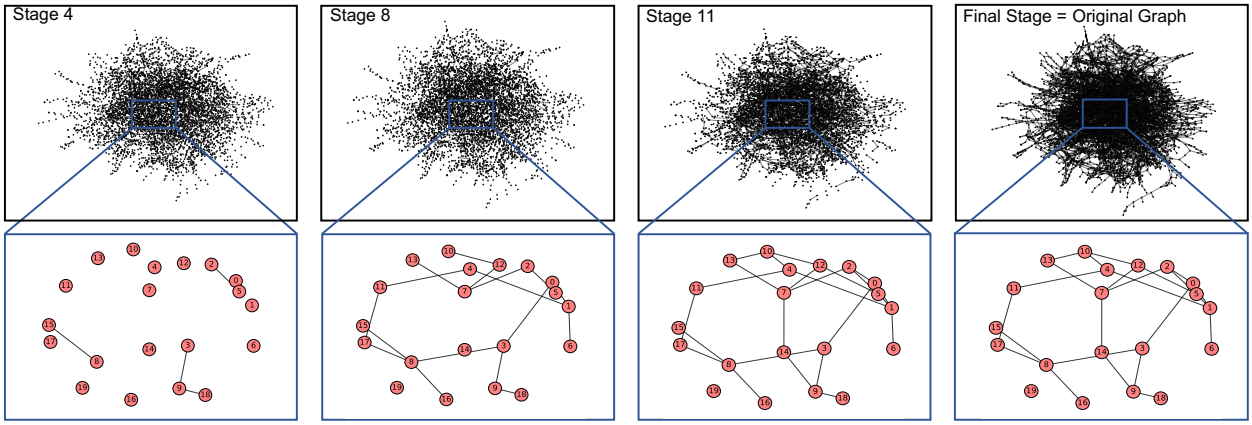


Figure 2.7: The topology of original graph and corresponding stages of road network.

Figure 2.8 and Figure 2.9 respectively show the log-log degree distributions and log-log cluster coefficient distributions for each of the datasets given in Table 2.1. In Figure 2.8 (2.9), the horizontal axis in each degree distribution represents the ordered degrees (ordered cluster coefficient), with the vertical axis representing the density of each degree (cluster coefficient). For each sub-figure, we use a red line to demonstrate the degree (cluster coefficient) distribution of the original graph, and use a set of colors (gradient from green to blue) to represent the corresponding distributions of the ordered stages.

We observe that except for the ER network, the degree distributions and the cluster coefficient distributions of early stages are similar to the original graphs, and only become more so as we progress through the reconstruction stages. Although the degree distributions and cluster coefficient distributions for the early stages of the ER network reconstruction are shifted, we observe that GTI quickly learns the Poisson like shape in degree distribution, and also learns the "peak-like" shape in the cluster coefficient distribution. This is particularly noteworthy given that the ER model has no true underlying structure (as graphs are chosen uniformly at random). Finally, we note that the cluster coefficient for each node in WS-1 is zero, and we have observed that GTI learns this feature very quickly, even in the first few stages. However, as we cannot take the log of zero in the subplot, we plot the behavior of GTI on WS-2 instead in Figure 2.9. The results show that in both cases (zero or non-zero cluster coefficient), GTI is able to learn the corresponding characteristics.

In addition, we also use Frobenius norm [98] (see Equation (2.4)) and average node-node similarity [99] (see Equation (2.5)) to evaluate the similarity between generated stages and the original graph. Let $A = G - G'_{L-1}$, where G'_{L-1} is the adjacency matrix of the penultimate stage. The

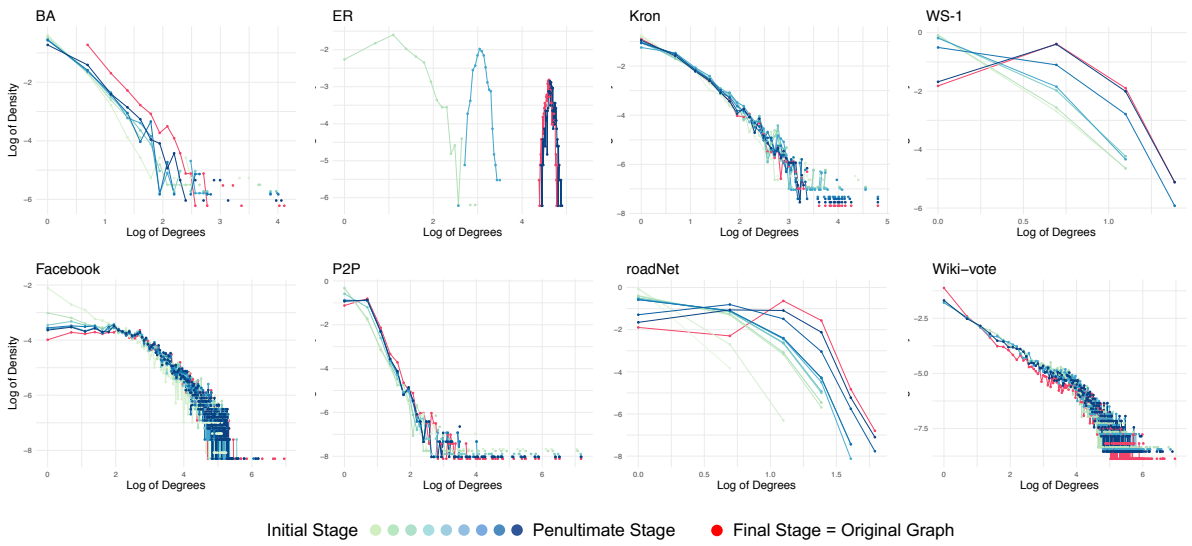


Figure 2.8: Degree distributions for 8 datasets.

notation A^T denotes the matrix transpose of A, and $sim\left(n_i^{G'_{L-1}}, n_j^G\right)$ represents the node-node similarity with mismatch penalty [100] between node $i \in G'_{L-1}$ and node $j \in G$. The total number of nodes in the original graph is given by $\left|N^G\right|$. Here, the F-norm calculates the distance between two adjacency matrices, and average node-node similarity indicates how G'_{L-1} resembles G.

$$\text{F-norm} = \sqrt{\sum_{i=1}^{N}\sum_{j=1}^{N} a_{ij}^2} = \sqrt{trace(A^T A)} \tag{2.4}$$

$$sim(G'_{L-1}, G) = \frac{\sum_{i \in N}\sum_{j \in N} sim\left(n_i^{G'_{L-1}}, n_j^G\right)}{|N^G|} \tag{2.5}$$

For comparison, we use the same model parameters to generate 100 ensembles of BA, ER, Kronecker and WS networks. These newly generated networks serve as base models to help us evaluate the similarity between the penultimate stage and the original graph. Table 2.2 and 2.3 respectively display the F-norm and the average node-node similarity between the penultimate stage and original graph. Here, bold letters imply best values between the penultimate stage and the mean value of corresponding base models for each dataset. We likewise generated 100 samples for each base model. Hence, for the *BaseModel* and *Penultimate Stage* Columns in Table 2.2 and Table 2.3, we add the standard error for each generated network.

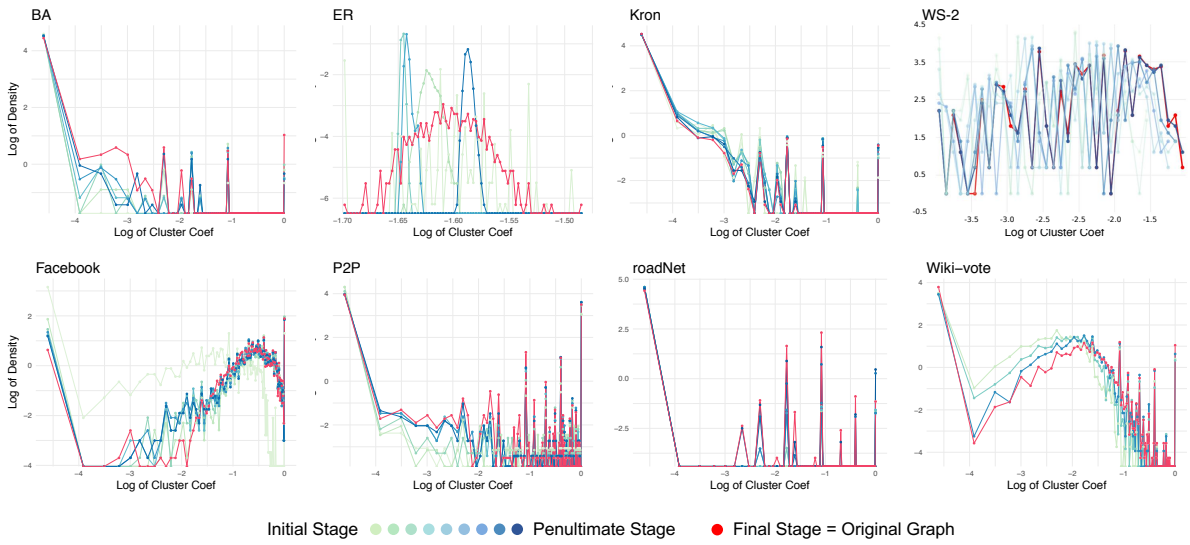


Figure 2.9: Cluster coefficient distributions for 8 datasets.

- Table 2.2 shows that 3 of 4 penultimate stages (i.e. BA, Kronecker, WS) have smaller F-norm distances, which means GTI successfully maintains the topological information of the original graph.
- Table 2.3 shows that the penultimate stage of BA has larger average node-node similarity with the original graph, and the penultimate stages of Kronecker and WS and the corresponding base model have identical similarity to the original graph. These results give a solid evidence that GTI also has the ability to attain good node-level similarity to the original graph.
- The ER network is a totally random graph model and hence GTI performs slightly worse than the base models (i.e. due to a lack of important topological structure for GTI to learn).

Table 2.2: F-norm distance numerical evaluation on penultimate stage and original graph

Graph	Penultimate Stage	BaseModel	BaseModel_{min}	BaseModel_{mean}	BaseModel_{max}
BA	**53.0283**±0.0436	62.9031±0.0859	62.8172	62.9031	62.9762
ER	285.8566±0.6287	282.9252±0.5637	282.3615	**282.9252**	283.4431
WS-1	**44.4522**±0.0372	44.6094±0.0448	44.5646	44.6094	44.6542
WS-2	**44.3101**±0.0218	44.5103±0.0463	44.4640	44.5103	44.5566
kronecker	**124.9320**±0.2698	125.323±0.2757	125.1987	125.1987	125.5987

Table 2.3: Average node-node similarity numerical evaluation on penultimate stage and original graph

Graph	Penultimate Stage	BaseModel	BaseModel_{min}	BaseModel_{mean}	BaseModel_{max}
BA	**99.9707%**±0.0047%	99.9640±0.0068%	99.9573%	99.9640%	99.9681%
ER	99.6372%±0.0196%	99.6590±0.0277%	99.6338%	**99.6590%**	99.6867%
WS-1	**99.9994%**±0.0001%	99.9994±0.0001%	99.9993%	**99.9994%**	99.9995%
WS-2	**99.9994%**±0.0001%	99.9994±0.0001%	99.9993%	**99.9994%**	99.9995%
kronecker	**99.2240%**±0.0593%	99.2241±0.0650%	99.1606%	**99.2240%**	99.2889%

2.5.4 Comparison with Graph Sampling

The graphs generated by GTI can be considered as samples of the original graph in the sense that they are representative subgraphs of a large input graph. We compare the performance of GTI with that of other widely used graph sampling algorithms (Random Walk, Forest Fire and Random Jump) [63] with respect to the ability to retain topological structures . We benchmark on the subgraph structures of the BA and Facebook datasets to compare the stage 1 of GTI against the graph sampling algorithms (designed to terminate with the same number of nodes as the GTI stage).

We show each of subgraphs from BA and Facebook networks (nodes 0-19 and nodes 0-49) to visually compare the ability of the first stage of GTI to retain topological features in comparison to the three graph sampling methods. In Figure 2.10, we observe that the stage 1 of GTI has retained a similar amount of structure in the 20 node BA subgraph as Forest Fire [64], while demonstrating considerably better retention than either Random Walk or Random Jump. However, for the 50 node BA subgraph, only GTI has the ability to retain the two super hubs present in the original graph. In Figure 2.11, we observe that GTI demonstrates vastly superior performance to the other methods on the Facebook dataset, which has a number of highly dense clusters with very sparse inter-cluster connections.

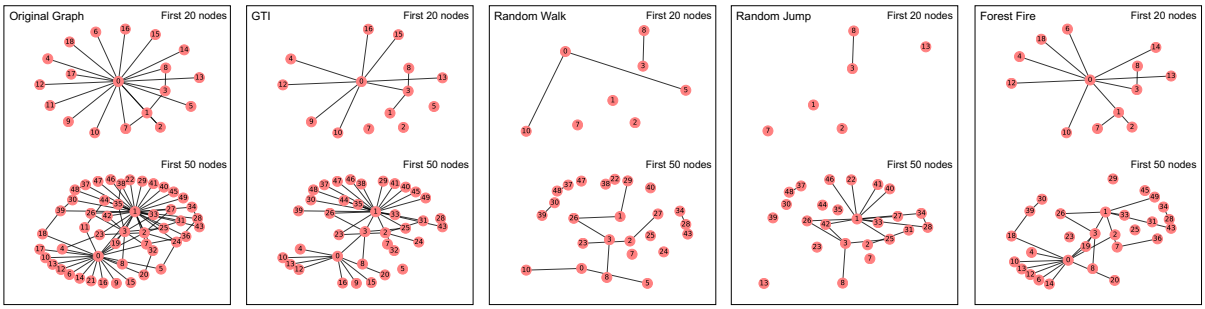


Figure 2.10: Comparison with graph sampling methods on the BA subgraphs.

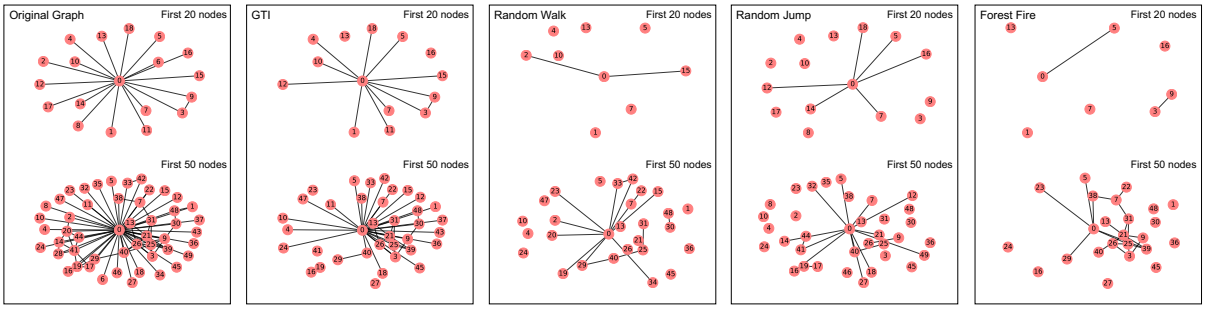


Figure 2.11: Comparison with graph sampling methods on the Facebook subgraphs.

2.6 Discussion

This chapter leveraged the success of GANs in (unsupervised) image generation to tackle a fundamental challenge in graph topology analysis: a model-agnostic approach for learning graph topological features. By using a GAN for each hierarchical layer of the graph, our method allows us to effectively reconstruct input graph, preserving both local and global topological features. In addition, our method is able to automatically learn the number of stages required to construct the graph non-parametrically. This is potentially advantageous in terms of understanding the distinct stages of graph reconstruction, and providing interpretable insight into the important features of the graph. Our experimental findings show promising results on the capability of GTI for learning distinct topological features from different graphs. To the best of our knowledge, there is no single graph model that can capture these distinct topological features.

The work completed for this chapter demonstrates the need for approaches capable of dealing with the wide variety of graph structures, as well as their potential. However, despite GTI's capability for dealing with many graph structures without the need for manual tuning, it still

fundamentally relies on many of the same tools for ingestion and processing as other deep learning methods (e.g. CNNs). In the next chapter, we begin to move towards approaches that "ingest" the graph in a more natural way; exploration and traversal.

Chapter 3

Graphs and Machine Learning for Decision Support

This chapter continues the discussion of traditional ways to input graphs to machine learning models, and introduces the DeepID, which can be viewed as our first step on the path to a method of natural and interpretable input of graphs to machine learning algorithms. The approach followed in this chapter differs somewhat from that of the more general Computational Graph Query outlined in Chapter 5 in that the DeepID explicitly encodes the graph structure into the machine learning model (though it nonetheless is a special case of the CGQ methodology). A key contribution of this chapter is in demonstrating the importance of interpretability in machine learning models, and how it enables us to easily incorporate targeted robustness constraints.

3.1 Traditional Approaches for Machine Learning with Graphs

As discussed briefly in Chapter 2, the traditional methods for solving deep learning problems involving graphs look much the same as the methods for static inputs. Rather than viewing the graph as a whole, the graphs is "summarized" in some manner. In industry, this will often involve manual computation and "feature selection" [101–103] where a graph is summarized according to known relevant properties. For example, in problems of money laundering detection on financial transaction networks [104–106], the presence of large cycles is often suspicious, and so graphs may be pre-processed to identify, count, and locate such cycles. This information (rather than the graph

itself), is then input to traditional machine learning approaches, as this summary information typically has the required static form (i.e. can be summarized by tensors) that graph objects themselves lack. An even more common summary tensor of the graph is the adjacency matrix; indeed, recall that in both Chapters 1 and 2, this was the primary method through which information about the graphs were input to the model. This information is then read as a simple tensor; machine learning algorithms in general have no way of knowing that a supplied matrix represents a graph structure as opposed to any other binary (or weighted, for a graph with defined edge-weights) matrix.

Of course, doing this process manually for every problem is not practical, and so many methods have been developed to automatically extract features, or project the graph onto a vector space (where we can view the coordinates of the embedded vector space, when given a graph, as features). This approach is typically referred to as "graph embedding". This approach is exemplified by one of the most prominent methodologies for performing machine learning on graphs, node2vec [107], where random walks are used to create a vector embedding for each *node* (as opposed to the entire graph), with these vectors then input as a representation of the graph. There are also whole graph embeddings [108] (typically used for quick classification problems involving varied smaller graphs, e.g. molecule identification [109]) and everything in between (for a comprehensive review, see [110–112]). Unfortunately, embeddings can be impractical in many situations. For instance, a given vector space may work well for a money laundering problem, but perform poorly when trying to predict non-performing loans [113].

We see this approach of embedding, in general, as trying to mold the problem to fit existing solutions, rather than expanding upon existing solutions to develop new methodologies that can instead interpret graph structures in a more natural way with more precise detail. In this dissertation, we accomplish this partly by mimicking how graphs are analysed in graph databases, through querying (see Chapter 5 for more detail on this perspective). This is essentially taking a lesson from graph databases and applying it to deep learning. Fortunately, this transfer of knowledge can go in multiple directions. In Chapter 1, we outlined techniques for discrete optimization via continuous approximation; in this chapter, we outline how these techniques (as well as other deep learning methodologies like GANs) can be applied to solve an important interpretable problem in graph analytics, that of using graphs to model and solve decision problems.

3.2 Introduction to Decision Support Systems

The goal of decision making frameworks is to provide a tool for representing key elements affecting agents' decisions and their relationships, and to provide assistance in selecting good decisions. In most decision making frameworks, there is a trade-off between interpretability, i.e. the ability to clearly identify the relationships between the key elements, and computational efficiency (and large memory requirements). There are many decision making settings, in particular, those involving financial or regulatory decisions, where interpretability often takes precedence over efficiency. In this chapter, we outline the DeepID, a neural network approximation of Influence Diagrams, that avoids *both* pitfalls. We demonstrate how this framework alsoallows for the introduction of robustness in a very transparent and interpretable manner, without increasing the complexity class of the decision problem.

Influence Diagrams (IDs) [114] were one of the earliest quantitative approaches for single agent decision making. IDs represent the elements relevant to a decision making problem using a directed graph consisting of three kinds of nodes (see Figure 3.1a): "oval" chance nodes that represent exogenous random variables, "rectangular" decision nodes where the decision maker chooses a strategy (i.e. a distribution over available actions), and one or more "rhombus" utility nodes that output a utility for the chosen strategy. Initially, IDs had one utility node, but this was later relaxed. The dependence between the exogenous random variables, the strategy, and the utility is encoded by directed edges. The objective in a (single-agent) ID is to compute the strategy that maximizes expected utility. An ID reduces to a Bayesian network once a strategy is chosen at each of the decision nodes, thus inheriting the conditional dependence structure of the Bayesian network. This allows for the relationships between key elements impacting the decision problem to be defined in a clearly interpretable manner. The ID in Figure 3.1 represents a decision problem with three decisions and two chance nodes. The conditional independence $C_2 \perp D_1 | D_2$ is clearly apparent from the ID representation.

Although IDs facilitate interpretable decision analysis, IDs are not able to efficiently represent and integrate over distributions associated with the chance and decision nodes. IDs typically require a "no-forgetting" condition where all future decisions must retain knowledge of, and thus in general, be dependent on all past decisions with complexity growing exponentially with network size. Consequently, IDs have been limited to relatively small decision problems with small discrete

chance and decision distributions, and only limited support for continuous distributions [115, 116]. As a result, they have lost prominence as a method of choice for decision analysis, which has led to a lack of new academic literature in the field (though still find some use in the medical domain [117] due to the particular importance of interpretability in the field).

In direct contrast to IDs, Markov Decision Processes (MDPs) and their extensions [118] must satisfy the Markov property wherein the state encodes all relevant past information, and the optimal action is only a function of the current state. Consequently, there is a possibility that the optimal policies can be computed efficiently. However, in many decision problems, the Markov property is achieved only by defining a very complex state such that the resulting MDP lacks any interpretability [119]. The graph in Figure 3.1b is an MDP representation for the ID on the left hand side. In order to create the MDP we have assumed that each chance node C_i, $i = 1, 2$, and decision node D_j, $j = 1, 2, 3$, has a binary outcome $\{0, 1\}$. The gray nodes are the states, and the white nodes are the allowed values of actions in a particular state. The MDP has a larger graphical representation because one has to track all past decisions in the state definitions to satisfy the Markov property; often making it impossible to characterize the optimal policy because of the curse of dimensionality. But perhaps more importantly, the conditional independence structure is completely obscured.

An increasingly important requirement for decision problem is robustness to uncertainty in the elements of the associated decision problems. The tension between computational efficiency and an interpretable representation becomes even more stark when one is uncertain about some of the parameters and wants to be robust to these perturbations. The decision analysis framework must allow agents to model uncertainty in a more granular manner, e.g. uncertainty in distribution of exogenous noise, or execution uncertainty, or uncertainty in the risk tolerance of the agent. For example, it is possible that the distribution of the chance node C_1 (see the ID on the left side of Figure 3.1) is uncertain; however, the distribution of the chance node C_2 has no error. We propose a framework that allows for the modeling of such targeted uncertainty; moreover, the resulting framework retains the macro-level structure of an ID, albeit with a few more nodes. Targeted uncertainty is very hard to introduce in MDP models for decision problems because the requirement to maintain the Markov property completely obscures the conditional independence structure of the decision process. The chance nodes C_1 and C_2 are conflated into the state S_1,

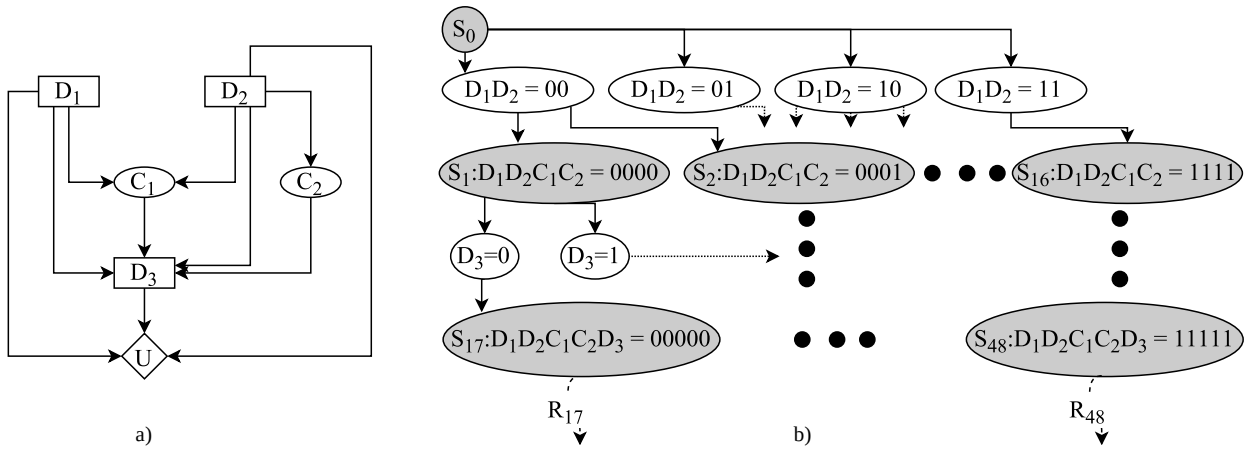


Figure 3.1: Example ID (a) and a corresponding MDP formulation (b).

since the outcomes of both these nodes define the decision D_3 (see the MDP on the right hand side of Figure 3.1). Thus, the targeted uncertainty in C_1 has to be represented as a structured uncertainty in the transition matrices. It is well-known that introducing structured uncertainty in the transition matrices is computationally intractable [120, 121].

To summarize, IDs are interpretable and allow for the targeted modeling of uncertainty; however, the difficulties associated with representing distributions and integrating over these distributions are prohibitive. Recent developments in the deep learning community precisely address this computational difficulty. Differentiable generator nets (DGNs) essentially convert the task of generating samples from (or equivalently, integrating over) complicated distributions into a function approximation task [122]. The function is represented by a deep neural network that can be efficiently trained using gradient descent. Generative adversarial networks (GANs) [69], have made the task of such a function approximation more efficient by removing the need to maintain a balance between the generator and the discriminator, and also reduced mode dropping. Variational auto-encoders [123] extensively employ DGNs to efficiently learn high dimensional posterior distributions. Conditional GANs [124], Wasserstein GANs [125] and autoregressive networks e.g. [126] extend function approximation to learning conditional distributions.

3.3 The DeepID

We propose a new representation for robust decision making called DeepID [4] that retains the Bayesian-like conditional independence interpretability of IDs, but does not suffer from the asso-

ciated computational difficulties. In our representation, each chance, decision, and utility node is represented by a DGN. Thus, selecting strategies in decision nodes reduces to optimizing over the parameters of a neural net. GANs can efficiently encode both a very large class of continuous distributions, as well as discrete distributions via Concrete distributions (see Chapter 1). We show that the DeepID framework allows us to introduce targeted uncertainty in a very flexible manner by appropriately introducing additional chance, decision, and utility nodes. In the DeepID framework, we model each node of an ID as a separate deep network, and connect the corresponding networks according to the macro-level structure of the original ID; thus, the larger deep network we construct retains the interpretability associated with IDs. From DGNs and GANs, we inherit computational efficiency, as we are able to use all the associated tools.

In the resulting network, the chance and utility nodes can be trained apriori as GANs since they approximate conditional distributions; whereas the decision nodes are collectively trained as feed-forward neural networks to optimize the sample mean. This is in contrast to traditional end-to-end feed-forward deep networks constructed by connecting a series of layers in an ad-hoc fashion, which lack interpretability.

We focus primarily on the case of a single agent in a one-shot game [127], which covers important contexts like medical decisions [128]. Robust decision making is particularly important in one-shot settings since agents do not have the ability to repeat the game or update models of uncertainty. Our main contribution in this chapter is to show that robustness can be introduced in a very flexible and interpretable manner in IDs, that can then be trained by constructing and optimizing the corresponding interpretable DGN. We propose DeepID as a tool for interpretable robust decision making. In the DeepID approach, the nodes of an ID are replaced by DGNs with free parameters for decision nodes, and fixed parameters for chance or utility nodes.

A DGN transforms the task of generating samples from (or, integrating over) a complex distribution into a function approximation task. Let $f(\mathbf{x}|\mathbf{y})$ be a given conditional distribution. A DGN generates samples from this distribution by first generating a sample $\boldsymbol{\epsilon}$ from a known distribution and transforming the sample using a differentiable nonlinear function $g_\theta(\mathbf{y}, \boldsymbol{\epsilon})$. Thus, the task of representing the distribution $f(\mathbf{x}|\mathbf{y})$ reduces to learning the parameter θ. This task is non-trivial; however, there is now a mature literature on how to efficiently and robustly learn the parameter

θ [69, 124]. In this work, we leverage this technology to argue the computational efficiency of DeepIDs.

Recall that an ID consists of set chance nodes C, decision nodes D, and utility nodes U, connected by directed arcs representing the conditional independence relations. In the DeepID framework, each chance node and utility node is replaced by a DGN, i.e. the function $g_{\bar{\theta}_c}(\pi(c), \boldsymbol{\epsilon})$, where $\pi(c)$ denotes the outputs at the direct parents of the node c in the ID representation (for example, in Figure 3.1 $\pi(C_2) = D_2$, and $\pi(U) = \{D_1, D_2, D_3\}$), the fixed parameter $\bar{\theta}_c$ is learned by matching conditional distributions using a GAN, and $\boldsymbol{\epsilon}$ are samples from a given fixed distribution. In contrast, at a decision d, the strategy is represented by the DGN $g_{\theta_d}(\pi(d), \boldsymbol{\epsilon})$ where the parameter θ is chosen to maximize the expected utility. Thus, DeepID decision nodes do not directly represent strategies over actions; rather, parameters of DGNs control the distribution of the generated samples. To the best of the authors' knowledge, this is the first ID approximation framework where decision nodes are replaced by a sample generating procedure. Some distributions (such as those in the location-scale family) can be easily implemented using DGNs. For example, suppose $f(\mathbf{x}|\mathbf{y}) = \mathcal{N}(\boldsymbol{\mu} + \mathbf{C}\mathbf{y}, \mathbf{R}\mathbf{R}^\top)$ is a multivariate Gaussian distribution with mean $\boldsymbol{\mu} + \mathbf{C}\mathbf{y}$ and covariance $\mathbf{R}\mathbf{R}^\top$. This distribution can be generated by the DGN $g_\theta(\mathbf{y}, \boldsymbol{\epsilon}) = \boldsymbol{\mu} + \mathbf{C}\mathbf{y} + \mathbf{R}\boldsymbol{\epsilon}$, where $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, which is a simple linear layer with an offset. As in Chapter 1, we can use this reparameterization trick [123] to create simple but exact chance nodes (where parameters are trained or set apriori) or decision nodes, where parameters are free to be trained during joint learning of the full DeepID DGN. Whenever a known differentiable reparameterization exists, we can significantly reduce the complexity of learning appropriate DGNs (and even eliminate it entirely, e.g. for a chance variable in the location-scale family with known mean and variance).

The DeepID reduces the ID into a interpretable DGN – differentiability is very important to ensure that the strategy space can be searched efficiently. In many applications of IDs, discrete distributions play a very important role. In order to be able to model these in the DeepID, we need a differentiable approximation of discrete distributions. As discussed in Chapter 1, the Concrete distribution allows us to approximate discrete distribution using DGNs. In the Concrete relaxation, recall that samples from a discrete set of size n are approximated by samples from the continuous simplex $\Delta^{n-1} = \{x \in \mathbb{R}^n | x_k \in [0, 1], \sum_{k=1}^n x_k = 1\}$, with vertices of the simplex being the one-hot vectors that can be mapped to the elements of the discrete set. Recall from Equation (1.2) that

α parameters representing unweighted probabilities across the categorical values determine the probability of a given vertex being the closest to the generated sample. The key feature here is that the parameter α is allowed to be a differentiable function of the parameters or outputs of the parent nodes in the DeepID. Thus, complex probability tables can be approximated as conditionally dependent functions of parent nodes that output the appropriate values of α, leading to a significant reduction in storage.

Clearly, the DeepID framework is of interest only if we one can guarantee that the optimal strategy can be represented by the associated DGN. We show that a large class of IDs can be arbitrarily closely approximated by DeepIDs, with optimal solutions that correspond to one another. Consider an ID with chance nodes C, decision nodes D and utility nodes U. Suppose the outputs of all nodes $i \in C \cup D$ take values in a compact set, the inverse conditional CDF $F_i^{-1}(x_i|\pi(i)\}$ is continuous for all $i \in C \cup D$, and the utility functions are differentiable and bounded.

Theorem 3.1. *(a) Let $\bar{\sigma} = \{\bar{\sigma}_d\}_{d \in D}$ denote any strategy profile across all decision nodes D in the ID, and let $\mathbb{P}_{\bar{\sigma}}$ denote the corresponding joint distribution over actions, chance and utility node outcomes. Then there exists a sequence of DGNs $g^{(n)}$ and parameter vectors $\theta^{(n)}$ such that the corresponding joint distribution over actions, chance, and utility outcomes $\mathbb{P}_{g^n_{\theta^{(n)}}} \xrightarrow{D} \mathbb{P}_{\bar{\sigma}}$.*

(b) Let σ^* denote the optimal strategy for the ID, with expected utility $\mathbb{E}[u(\sigma^*)]$. Then there exists a sequence of DGNs g^n with input size m such that $\mathbb{E}[u(g^n_{\theta^{(n)}_{\max}}(X))] \to \mathbb{E}[u(\sigma^*)]$, for $X \sim$ Uniform$[0,1]^m$, where the components of the parameter vector $\theta^{\max}$ corresponding to the chance and utility nodes are defined by matching conditional distributions apriori, and the parameters corresponding the decision nodes are computed by the maximization of the expected utility.

Proof. (a) We first prove the result for an ID consisting of a single multivariate decision node d with no parent nodes, where the decision involves selecting a vector a from a set $A \in \mathbb{R}^m$, with the utility defined as the sum of the components of the decision node vector. We then argue that any ID satisfying the theorem conditions can be cast into this setting, proving the result for IDs with multiple chance, decision, and utility nodes.

Let $\bar{\sigma}_d$ denote the random strategy at node d. Let $F_i(x_i|a_1, a_2, \ldots, a_{i-1})$ denote the inverse conditional CDFs for the i-th component a_i of the decision vector conditioned on the previous $i-1$ components: $a_1, \ldots, a_{i-1}$. We assume that F_i is continuous in $(x_i, a_1, \ldots, a_{i-1})$.

Define a function $g : [0,1]^m \to \mathbb{R}^{m+1}$ as follows:

$$\begin{aligned} g_1(x) &= F_1(x_1) \\ g_k(x) &= F_k\big(u_k \mid g_1(x), \ldots, g_k(x)\big), \quad k = 2, \ldots, m. \\ g_{m+1}(x) &= \textstyle\sum_{k=1}^{m} g_k(x) \end{aligned}$$

Let $X \sim \text{Uniform}[0,1]^m$. Then it is clear that $g(X) \stackrel{D}{=} \mathbb{P}_{\sigma_d}$. Since each conditional inverse CDF F_i is continuous in all its arguments, the function g is also.Therefore, by the universal approximation theorem for feed-forward neural networks [129], for $n \geq 1$ there exists a network $g^n_\theta : [0,1]^m \to \mathbb{R}^{m+1}$ and parameter value θ^n such that $\sup_{u\in[0,1]^m} |g^n_{\theta^n,k}(x) - g_k(x)| \leq \frac{1}{n}$ for all $k = 1, \ldots, m+1$. Thus, $\lim_{n\to\infty} g^n_{\theta^n} = g$, where the convergence is uniform. Next, uniform convergence of the sequence of functions implies that $g^n_{\theta^n}(X) \stackrel{D}{\Rightarrow} g(X) \stackrel{D}{=} \sigma_d$.

Now consider a more general ID satisfying the assumptions of the theorem. A utility node is simply a chance node with a deterministic output, and a chance node is a decision node where there is only one choice. Any multi-node ID satisfying the conditions of the theorem can be cast as a single decision node over a vector, ordered according to a topological sort of the ID graph. That is, we view all utility nodes as chance nodes, and all chances nodes (including those that are actually utility nodes) as decision nodes. With a network consisting of nothing but "decisions", we can define a vector of "decisions" that corresponds to the "decisions" at each node. We can then concatenate all of these decisions as a single decision node over this vector (where only well-defined combinations of decisions are defined). At this point, the above argument can be applied, and the result is proved.

(b) Since σ^* is a given fixed randomized strategy, from (a) it follows that there exist a sequence of networks and parameters such that $g^n_{\theta^n}(X) \stackrel{D}{\Rightarrow} \sigma^*$.

The parameter $\theta^n_{\max}$ is defined as follows. Fix the parameters for the chance and utility nodes to the appropriate components of the sequence θ^n. Compute the components for decision nodes by maximizing the expected utility. Thus, it follows that $\mathbb{E}[u(g^n_{\theta^{\max}}(X))] \geq$

$\mathbb{E}[u(g^n_{\theta^n}(X))]$. Moreover, since $\theta^n_{\max}$ implements a randomized policy, it follows that we have $\mathbb{E}[u(g^n_{\theta^{\max}}(X))] \leq \mathbb{E}[u(\sigma^*)]$.

Since $g^n_{\theta^n}(X) \stackrel{D}{\Rightarrow} \sigma^*$, by the uniform integrability of the sequence of generator network functions it follows that $\mathbb{E}[u(g^n_{\theta^n}(X))] \rightarrow \mathbb{E}[u(\sigma^*)]$. Since $\mathbb{E}[u(g^n_{\theta^n}(X))] \leq \mathbb{E}[u(g^n_{\theta^{\max}}(X))] \leq \mathbb{E}[u(\sigma^*)]$, it follows that $\mathbb{E}[u(g^n_{\theta^{\max}}(X))] \rightarrow \mathbb{E}[u(\sigma^*)]$, and the result is proved.

□

3.4 Interpretable Robustness

Parameter and distributional uncertainty has been a focus of research in a number of different fields, with numerous techniques proposed for ensuring that the solutions are robust to the underlying uncertainties. Interpretability in the DeepID facilitates more than just network simplification and an ability to encode an understanding of the decision system. It also facilitates robust decision making that is itself interpretable. In this context, we define interpretable robustness to mean that users can target their uncertainty and risk return preferences to accomplish specific robustness goals by making the individual component networks robust in particular ways. In neural network training, one employs drop-out techniques, or adds noise to network parameters or the input data to ensure that the parameters converge to values that ensure robustness with respect to perturbations [122]. This goal can also be achieved by suitably regularizing the network parameters. Robustness in many problems is achieved by explicitly modeling the uncertainty as part of the problem. In the financial risk management context, coherent risk measures [130] ensure robustness to uncertainty in the distributions of the underlying risk factors. Conditional value at risk (CVaR) is a very popular coherent risk measure that ensures robustness by reweighting the worst quantile of losses. Clearly, robustness to uncertainty is equally important in decision problems. Decision problems, typically, involve a complex set of interacting elements, some more uncertain than others. Thus, it is beneficial for decision making frameworks to be flexible enough to allow for the targeted introduction of robustness.

In this section, we show that DeepIDs allow for targeted robustness to be introduced interpretably. In a DeepID, the purpose of each component deep network is retained; therefore, one can separately control the robustness of each component network. Introducing such targeted robustness

is nearly impossible in a standard deep network where we have relatively limited understanding of the function of particular nodes or sub-networks. MDPs also do not easily accommodate targeted robustness since the definition of state, and corresponding transitions, often obscure the underlying independence relations. State aggregation further makes such a task difficult. DeepIDs allow us to model the following different classes of uncertainties.

3.4.1 Distributional Uncertainty for Specific Chance Nodes

For a chance node $c \in C$ with the DGN $g_{\bar{\theta}_c}(\pi(c), \boldsymbol{\epsilon})$ we can modify the parameters $\bar{\theta}_c$ or the exogenous samples $\boldsymbol{\epsilon}$ to model distributional uncertainty. For example, consider the case where the network $g_{\bar{\theta}_c}(\pi(c), \boldsymbol{\epsilon}) = \mu_c + \sigma_c \epsilon_1$, where $\bar{\theta}_c = (\mu_c, \sigma_c)$ and ϵ_1 is distributed according to a location-scale family. Then we can encode an agents uncertainty in the mean of the output of node c by setting $\mu_c \leftarrow \mu_c + \gamma_c \epsilon_2$, where ϵ_2 is sampled from another zero mean density. This is represented graphically by adding a new chance node c' (representing ϵ_2), adding a directed arc (c', c), and updating $g_{\bar{\theta}_c}(\pi(c), \boldsymbol{\epsilon})$ accordingly.

3.4.2 Regularization for Specific Decisions

Consider an agents decision node d with the DGN $g_{\theta_d}(\pi(d), \boldsymbol{\epsilon})$, we can regularize the parameters θ_d to encourage certain properties at node d. A noteworthy application of decision level regularizers is to encourage particular discrete decisions to have pure or mixed strategies. This can be achieved by adding a utility node u with $g_{\bar{\theta}_u}(g_{\theta_d}(\pi(d), \boldsymbol{\epsilon}), \boldsymbol{\epsilon}')$ a p-norm regularization penalty on θ_d (see Section 3.5 for an example employing the Concrete distribution).

3.4.3 Execution Uncertainty for Specific Decisions

For an agents decision node d with DGN $g_{\theta_d}(\pi(d), \boldsymbol{\epsilon})$, we can add noise at any level – to θ_d, to $\boldsymbol{\epsilon}$, or the output of d – to encourage gradient descent to compute a stable decision strategy. We can also interpret this as encoding that decision execution isn't exact, with the agent sometimes making mistakes (e.g. a financial trading strategy where executing a trade at a desired price is not possible, or takes time). Introducing such an uncertainty will ensure that the chosen strategy is less sensitive to execution errors.

3.4.4 Custom Risk-Tolerances

We can modify the network $g_{\theta_u}(\pi(u), \boldsymbol{\epsilon})$ at a utility node u by adding a new layer that represents an agents' risk-reward tolerance, i.e. setting the output to $g_{\theta_{u'}}(g_{\theta_u}(\pi(u), \boldsymbol{\epsilon}), \boldsymbol{\epsilon}')$. This is represented by introducing a new node u' and adding an arc (u, u').

Note that all these modifications to a DeepID are equivalent to adding new chance or utility nodes, changing the objective of the training algorithm. Consequently, such modifications do not increase the complexity class of the problem.

3.5 Experimental Results

3.5.1 Robustness

Here we demonstrate how interpretably robust modifications of an ID can be reformulated as DeepIDs. For purposes of demonstrating this translation, we use a relatively simple ID, and straight-forward DGN formulations for all distributions. We focus on the Reactor Problem [131] that involves an agent choosing between constructing a conventional or an advanced reactor. The performance of both the conventional and advanced reactors are uncertain; however, the performance of the advanced reactor is less certain. The state of the reactor in the future takes one of the following three values: **C**atastrophic failure, **S**mall failure, and **N**o failure. The probability distributions for the future states, and the corresponding utilities are in Table 3.2. The agent conducts a test to predict the performance of the advanced reactor. The conditional probability table for the test outcome is given in Table 3.1.

The DeepID formulation for this problem is displayed in Figure 3.2, with the macro-level ID structure shown on the LHS in (a), and example DGN internal structure shown on the RHS in (b) (where shaded nodes are inputs or outputs, and non-shaded nodes are operations). It consists of 3 chance nodes, Conventional Reactor (CR), Advanced Reactor (AR) and Test Results (TR), 1 decision node, Reactor Choice (RC), and 1 utility node, Standard Utility (SU). Note that the SU node is not a rhombus. This is because it can be fed into a CVaR utility node (CU) to add robustness to the utility outcome. The RC decision node chooses between constructing a conventional or advanced reactor as a function of the output from the TR chance node that take three possible values: **C**atastrophic failure, **S**mall failure, and **N**o failure. The utility of the decision

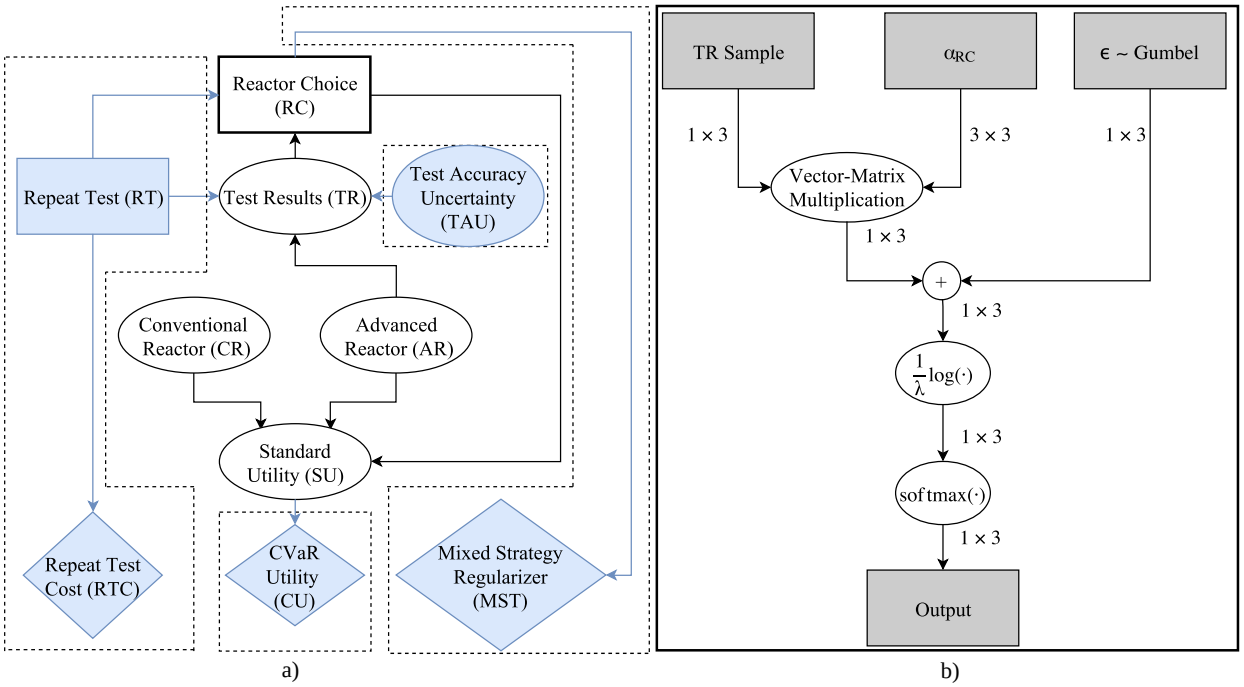


Figure 3.2: Reactor problem DeepID and modifications (a), and internal DGN of RC (b).

at RC node is a function of the true future state of the conventional and advanced reactors, and hence, the SU node has inputs from the CR, AR and RC nodes.

Since the outcomes of all nodes of the ID are discrete, we model the associated discrete distributions using the Concrete DGN as outlined in Section 3.4, with distinct $\alpha_{i|\pi(i)} \forall i \in C \cup D$ parameters for each possible combination of input and output. We let α_i denote the matrix of parameters that define the distribution at node i. The α parameters for chance nodes are fixed to the corresponding values, e.g. $\alpha_{TR=\mathbf{N}|AR=\mathbf{S}} = 0.288$), whereas the α corresponding to the decision node is a free parameter. For a chance node $c \in C$ (resp. decision node $d \in D$) with parent outcome $\pi(c)$ (resp. $\pi(d)$), a sample of the outcome from c is drawn according to the Concrete DGN using $\alpha = \pi(c)\alpha_c$ (see Figure 3.2b). Though not necessary for a problem of this size, we again note that the function approximation literature referenced in Section 3.2 can be used to let $\alpha = f(\pi(c))$ for some approximating function f when the size of α_c would be prohibitive.

Thus far we have described the standard DeepID for the Reactor Problem. Next, we demonstrate how to introduce robustness to the standard DeepID in an easily interpretable fashion. Each (optional) change is represented by additional shaded nodes and lines in Figure 3.2a. In our exper-

Table 3.1: CPT of test results given future advanced reactor performance

Future Performance	Test Results		
	Catastrophic failure	**S**mall failure	**N**o failure
Catastrophic failure	0.9	0.0	0.1
Small failure	0.147	0.565	0.288
No failure	0.0	0.182	0.818

Table 3.2: Reactor performance probabilities and associated utilities

Reactor	**C**atastrophic failure		**S**mall failure		**N**o failure	
	Prob.	Utility	Prob.	Utility	Prob	Utility
Advanced	0.14	-50	0.2	- 6	0.66	12
Conventional	N/A	N/A	0.02	-4	0.98	8

iments, we explore the effects of making each robust addition separately. We can, of course, also mix and match combinations of these robustness concepts.

CVaR Utility

We can replicate a CVaR like custom risk-tolerance by taking advantage of the sample-based nature of the DeepID. To set $CVaR_p$ as the objective function of the DeepID, we simply sort the output samples and take the mean of the pth quantile. This corresponds to a deterministic utility node with the function $g_{\theta_{u'}} = \frac{1}{k}\sum_{i=1}^{k} g_{\theta_u}(\pi(u), \epsilon_{[i]})$ where $k = \lfloor pN \rfloor$ and $g_{\theta_u}(\pi(u), \epsilon_{[i]})$ is the i^{th} smallest standard utility sample calculated through a sort function. Note that although $g_{\theta_{u'}}$ is in general not differentiable, there exist standard techniques to smooth this function to any degree of accuracy [132]. We demonstrate how this affects the DeepID layout in Figure 3.2a, where the regular utility node now feeds into the CU node at the output. This change was incorporated at the end of the network where we have a clear concept of good and bad utilities, though such a sample-based transformation could be applied anywhere in the network.

Uncertainty in Test Accuracy

In the standard formulation, we are quite sure about the ability of the test to correctly predict a catastrophic failure, and so we choose to build the advanced reactor if the test result is good. However, we may be interested in observing how our suggested behavior changes if we introduce distributional uncertainty to our test accuracy. In this experiment, we add $Q \sim \text{Uniform}(0, a)$ noise

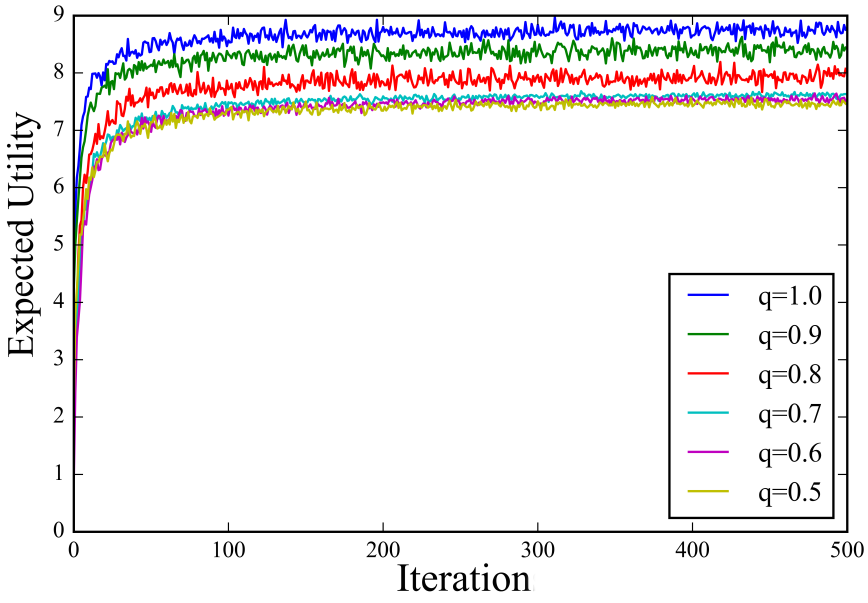


Figure 3.3: Training as $CVaR(p)$ is decreasing in p.

as $\alpha_{TR=\mathbf{C}|AR=\mathbf{C}} \leftarrow \alpha_{TR=\mathbf{C}|AR=\mathbf{C}} - Q$ and $\alpha_{TR=\mathbf{N}|AR=\mathbf{C}} \leftarrow \alpha_{TR=\mathbf{N}|AR=\mathbf{C}} + Q$ for increasing values of a. In Figure 3.2a, this corresponds to the additional chance node TAU representing the noise Q that is fed into $\alpha_{TR|AR}$.

Test Repetition

In some decision problems, we may have the ability to repeat actions to improve certainty. For instance, we may choose to repeat the advanced reactor test for some additional cost, resulting in an overall testing procedure that is more accurate. We show in Figure 3.2a how a new node can be added to represent this ability. The new decision node is connected to both TR (as it changes the accuracy of the test to be more accurate) and RC (because we must know when making the decision if our test results correspond to the more accurate situation where we have run an additional test), changing the α_i matrices accordingly. We experiment with this modification for increasing test costs (tc), with the additional test cost corresponding to a new utility node. Note that in this experiment, TR uses the conditional probability table of Table 3.3, where the Catastrophic failure prediction corresponds to the probability of *any* individual test in the overall testing procedure suggesting Catastrophic failure.

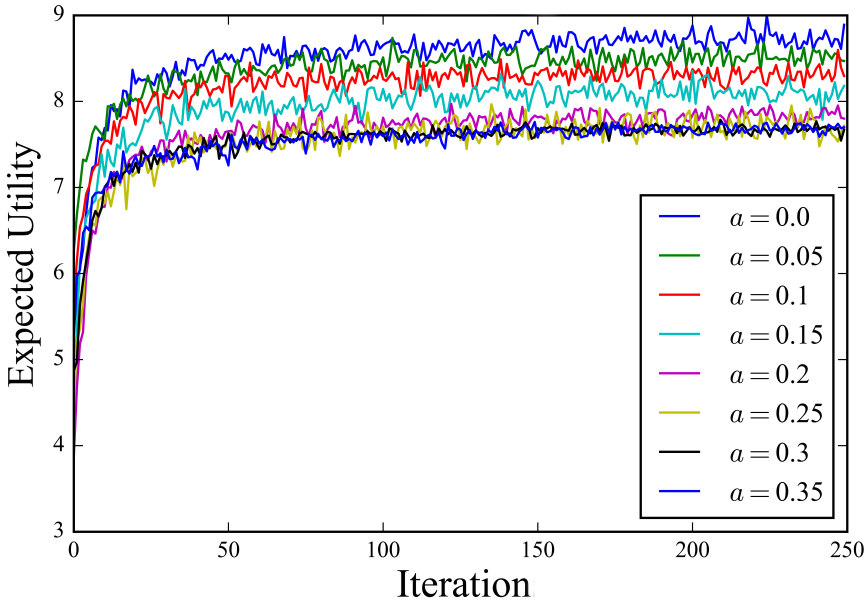


Figure 3.4: Training as Test Accuracy is Reduced.

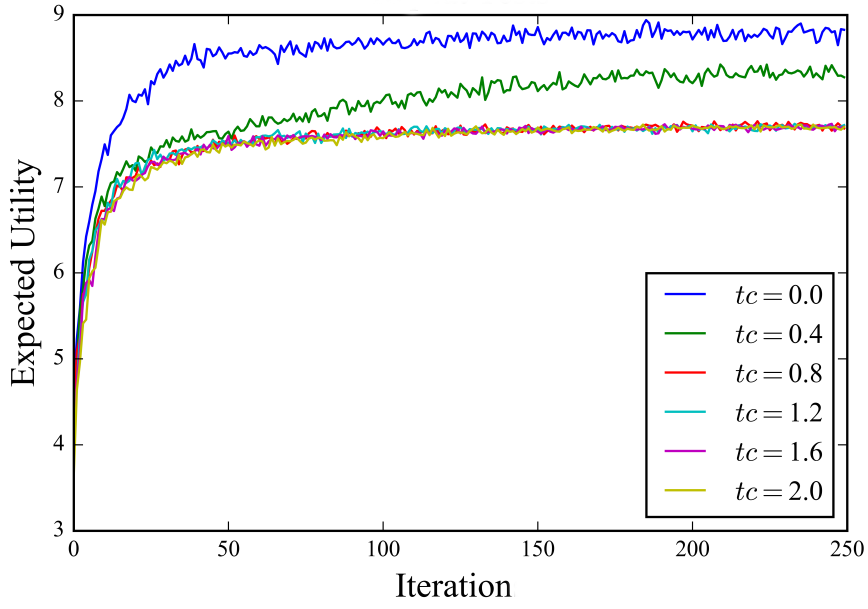


Figure 3.5: Training with different costs to repeating a test.

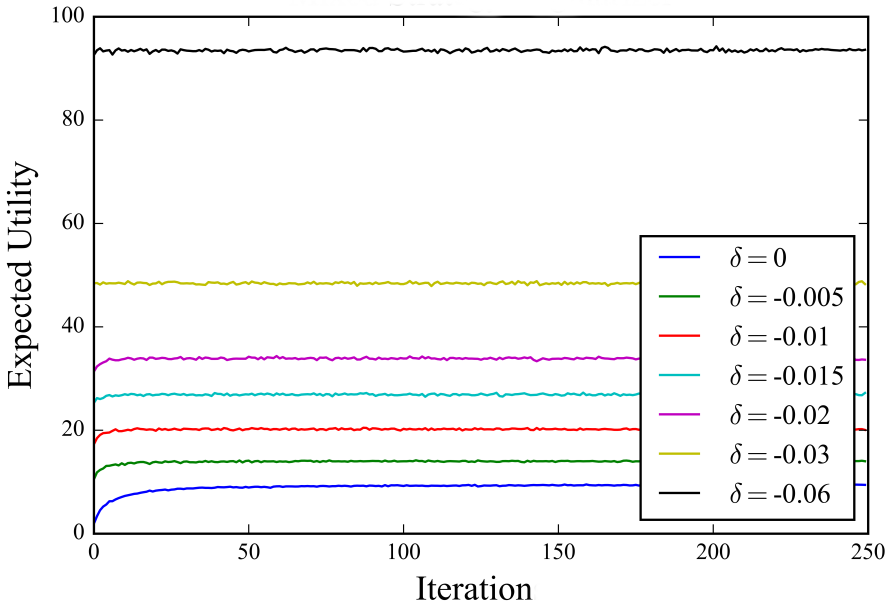


Figure 3.6: Training with different regularization strength.

Table 3.3: Test Results CPT given future catastrophic performance and repeat testing modification

Num. tests	Test Results | C		
	Catastrophic failure	**S**mall failure	**N**o failure
1	0.7	0	0.3
2	0.91	0	0.09

Mixed Strategy Regularization

In the standard ID formulation of the reactor problem, learning encourages the use of pure strategies. However, we may have a preference for mixed strategies in certain situations. Here, we demonstrate how regularizers can be added to individual decisions to encourage or discourage mixed strategies for particular decisions. We add a p-norm regularizer on α_{RC} through a new utility node MST with $g_{\theta_{MST}}(\pi(MST)) = \delta \left\|\alpha_{RC|TR=\mathbf{C}}\right\|_p + \delta \left\|\alpha_{RC|TR=\mathbf{S}}\right\|_p + \delta \left\|\alpha_{RC|TR=\mathbf{N}}\right\|_p$ for $p = 0.1$, where we can force RC to have a mixed strategy or pure strategy by varying the parameter δ.

3.5.2 Scalability

Here we apply the DeepID approach to a series of IDs with an increasing number of parameters, allowing us to investigate some scaling properties of the method. With an eye towards applying

Table 3.4: Global Connection Game running times in (s) on Intel(R) Core(TM) i7-4700MQ @ 2.40GHz with 200 samples for 2000 iterations.

Distribution	$H = 12$	$H = 80$	$H = 572$	$H = 4352$
Concrete	3599	5325	9247	28473

the DeepID framework in large multi-agent systems, we define a DeepID for the special case of the Global Connection Game [133] where a central planner makes the decisions for all players (thus it is not truly a multi-agent system, though it is suggestive of that capability). In the Global Connection Game, each of $p \in \{1, \cdots, P\}$ players has a source node s_p and a sink node t_p in a directed graph $Z = (V, E)$ that they wish to have connected. In our Global Connection DeepID, the central planner wishes to find the cheapest path for each player such that the total cost of making all paths is as low as possible. We formulate the cost as the sum of the costs assigned to each player, using a mechanism where edge costs $c_e = 1 \ \forall e \in E$ are split between each player who has that edge in their path. Further, we restrict the graph Z to be a sequence of four fully connected layers, each with P nodes such that the source and sink nodes of each player p are the p^{th} nodes in the first and final layers, respectively. This graph is shown in Figure 3.7. By only including the costs of edges between the 2^{nd} and 3^{rd} layers (because unique sources and sinks for each player mean no edges can be shared by multiple players between the 1^{st} and 2^{nd}, and 3^{rd} and 4^{th} layers) as our utility, the optimal solution is clearly for every player to share the same edge between the 2^{nd} and 3^{rd} layers, for an optimal expected utility of 1.

This corresponds to an ID with no chance nodes, where the central planner must make decisions for each player in each layer about which node to connect to in the next layer, where each decision is a conditional distribution depending on a player's location in the current layer. We model the discrete distribution of each players' next destination in a DeepID formulation using P^2 variables (a simplex over P possible destinations for each of the P possible current locations that the decision is conditionally dependent upon). With four node layers and deterministic start and end points for each of the P players, the central planner has control over $H = P^2(1 + P)$ parameters in total. The DeepID formulation of this problem is shown in Figure 3.7b. In our results here (See Figure 3.8 and Table 3.4), we perform gradient updates using the FTRL Algorithm [134].

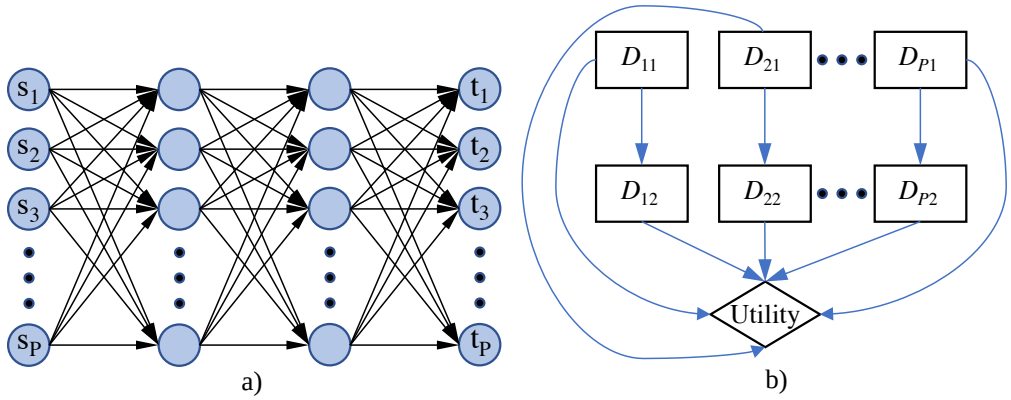


Figure 3.7: Global Connection Game graph for P players with designated source(s) and sink(t) nodes (a). Global Connection Game ID for P players where D_{lp} is the decision made in layer l for player p, where there is no third decision because players must terminate at the corresponding sink node (b).

Table 3.5: Experiment results summary

Test	Parameter	Strategy Before	Strategy After	Switchpoint
CVaR	dec. p	$BAoN$	ABC	0.7
Test Accuracy	inc. a	$BAoN$	ABC	0.3
Rep. Tests	inc. tc	Two tests, $BAoN$	Single test, ABC	0.8
MS Reg.	dec. δ	$\delta = -0.01, \alpha_{RC\|TR} = \begin{bmatrix} 0.93 & 0.07 \\ 0.55 & 0.45 \\ 0.41 & 0.59 \end{bmatrix}$	$\delta = -0.06, \alpha_{RC\|TR} = \begin{bmatrix} 0.65 & 0.35 \\ 0.51 & 0.49 \\ 0.49 & 0.51 \end{bmatrix}$	N/A

3.6 Results

The effects of our ID modifications in Section 3.5 are summarized in Table 3.5, where $BAoN$ is shorthand for the strategy "Build Advanced on a No failure test result, otherwise build Conventional", and ABC is shorthand for the strategy "Always Build Conventional". We observed that the convergence was not sensitive to the particular choice of algorithm or the learning rate. We note in Figure 3.3 with the CVaR modification that lowering p causes the learned strategy to change from $BAoN$ to ABC. With the advanced reactor having terrible consequences for failure, and a higher chance of failure, such samples dominate when the strategy allows for the advanced reactor to be built (due to imperfections in the testing procedure). As p is decreased, such samples increasingly dominate the utility. In contrast, ABC means we are indifferent to test inaccuracy, and $CVaR_p$ is relatively constant in p due to the small failure rate of 0.02. A similar trade-off is observed in Figure 3.4 when increasing a for $Q \sim \text{Uniform}(0, a)$. As a increases, the test is considered increasingly unreliable until the recommended strategy likewise becomes ABC. Though the result is ultimately

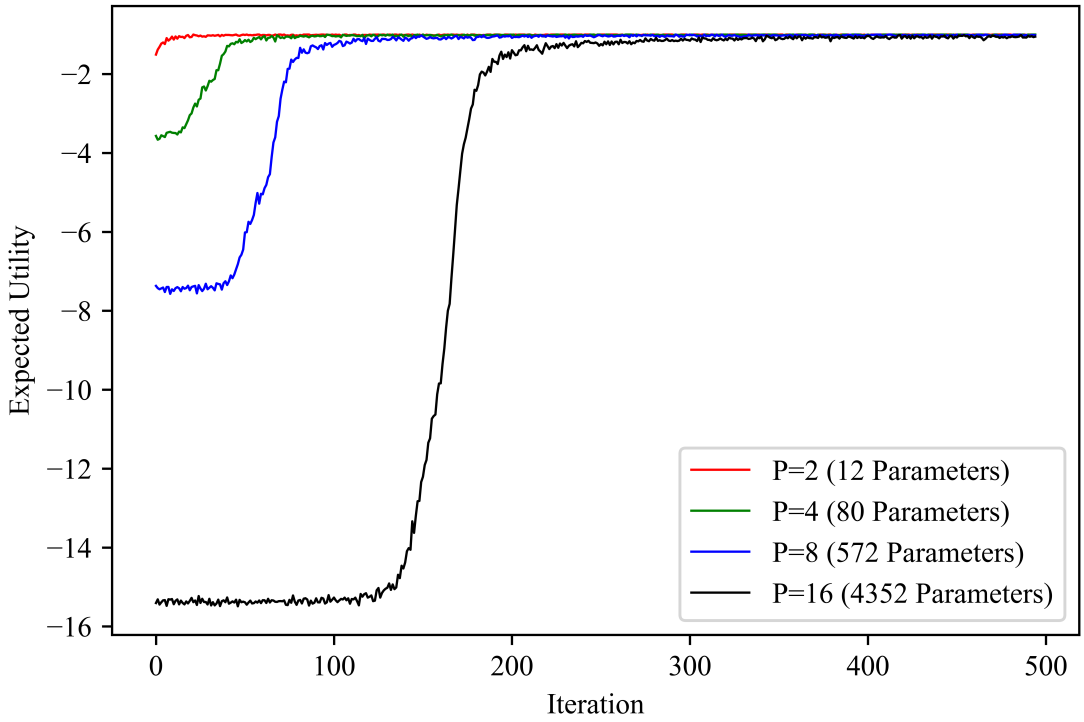


Figure 3.8: Expected utility training for FTRL (learning rate of 1.0) in the Global Connection Game, using $\lambda = 0.1$ and with all distributions initialized as uniform across all possible choices.

similar, the mechanisms of robustness that these tests represent differ, with CVaR representing risk-tolerance, and Q representing distributional uncertainty in a specific component of the model. As we increase tc in the RT modification (see Figure 3.5), we switch from being willing to pay for increased accuracy to allow the $BAoN$ strategy to be effective, until the increase in cost is greater than the difference between the $BAoN$ and ABC strategies. With the mixed strategy regularizer shown in Figure 3.6, we observe the same standard recommendation for $BAoN$, with the pure strategy suggestion becoming increasingly mixed as δ becomes more negative. Though it is not necessary in this example, we could also add a regularizer with $\delta > 0$ to encourage a pure strategy.

In Figure 3.8 and Table 3.4, we observe that the method displays favorable scalability properties for our toy Global Connection Game, with iteration time and iterations to convergence (to the correct value, as demonstrated by Figure 3.8) both demonstrating near-linear performance (for the number of parameters, and the number of players, respectively).

3.7 Discussion

This chapter presented an existence proof for the arbitrary accurateness of the DeepID and its ability to be trained with gradient based methods. As we will see in Chapter 5, this chapter was critical in showing how essentially separated fields such as graph based methods for decision making and machine learning can have synergistic capabilities when methodologies combining their properties are developed. We take this approach further in the next chapter. We also demonstrate how many of the techniques and approaches used in this chapter are required for a larger methodological breakthrough in Chapter 5.

Chapter 4

The SmartGraph

Graph databases and distributed graph computing systems have traditionally abstracted the design and execution of algorithms by encouraging users to take the perspective of lone graph objects, like vertices and edges. In this chapter, we introduce the SmartGraph [5], a graph database that instead relies upon thinking like a smarter device often found in real-life computer networks, the router. Unlike existing methodologies that work at the subgraph level, the SmartGraph is implemented as a network of AI enabled Communicating Sequential Processes (CSPs). The primarily goal of this design is to give each "router" a large degree of autonomy. We demonstrate how this design facilitates the formulation and solution of an optimization problem which we refer to as the "router representation problem", wherein each router selects a beneficial graph data structure according to its individual requirements (including its local data structure, and the operations requested of it). We demonstrate a solution to the router representation problem wherein the combinatorial global optimization problem with exponential complexity is reduced to a series of linear problems locally solvable by each AI router. This design is currently involved in two pending patents [135, 136]. This system also implements many of the functionalities mentioned in all chapters of this dissertation, as it is the SmartGraph that brings together many central tenets of this systems dissertation. Where appropriate, this functionality is covered in more detail in the associated chapters. This is particularly true of the Computational Graph Query outlined in Chapter 5, which is implemented in (and demonstrably benefits from many of the design elements of) the SmartGraph system, but is a novel contribution in its own right (to both graph querying, and in performing machine learning that requires the processing of graphs) so as to warrant a separate chapter.

4.1 System Concepts

4.1.1 Introduction to Graph Computing and Graph Databases

This chapter combines many of the lessons from graph computing with those of graph database design. This is a more common combination of fields than occurs in other chapters of this dissertation (e.g. graph databases and deep learning), primarily because Online Analytical Processing (OLAP) and Online Transactional Processing (OLTP), which refer to a focus on analytics and transaction response respectively [137], are typically seen as part of a "continuum". This is true for standard database and analytics technologies, as well as specifically for graph databases (which typically focus on OLTP; with functionality like storage and querying falling under the transactional domain) and graph computing (focusing on OLAP, where storage and the data itself is viewed as less important than the results of efficient large-scale calculations using that graph structured data) [138].

Data is often thought of in the context of input and output, to be used or analyzed by some external program or process. The structure of graph data, however, can indicate useful information about how to best execute graph based algorithms on that structure. This is demonstrated by a key refrain for existing graph computing paradigms, such as Pregel [139]: to "think like a vertex" [140]. In this work, we demonstrate the benefits of further integrating graph data with the analytics run on that data by creating an AI based graph database. When graphs have knowledge of their own properties, have the ability to send messages to other graphs, run calculations *concurrently*, and perform self-modification, a graph is no longer a static source of data. Instead it begins to resemble a network of routers. In this work we replace the "think like a vertex" mantra with "think *like a router*", using a router inspired abstraction to create a "SmartGraph" database. The method distinguishes itself from existing graph databases because of the AI enabled router abstraction; the routers, that are defined by the subgraphs they encapsulate, manage graph representation, concurrency and execution of operations *themselves*, as opposed to being simple static data managed by an external process.

Our work is not the first to combine OLAP and OLTP functionality. Indeed, there are many such works [137, 141–145] that claim to fall somewhere on the OLAP-OLTP "continuum". It is our perspective that the "continuum" as , typically, thought of (where to move towards an OLAP

design is to move away from a pure OLTP design) is not precisely correct. This is because of a simple premise; transactions can be used to execute analytics, if those transactions are properly managed. The difficulty, then, lies in properly managing small transactions such that the overall result of executing these transactions is to execute an analytic.

We do not claim to have been the first to have this view of the continuum in the graph sphere (even if it has not been explicitly stated in the academic literature as such). For example, Gremlin [146] executes queries on both OLAP and OLTP systems (Gremlin can be built on top of any graph system that implements its API) by sending "traverser" objects around a graph. However, as the years have passed, Gremlin has added additional functionality such that these "traversers" can also execute some simple graph analytics (e.g. PageRank [147]). There are of course many differences between Gremlin and our system; the most apparent of which is that Gremlin is a query language built on top of other systems (and not a system, in and of, itself). In addition, our system implements operations (some of which are described in Chapter 5) that go far beyond querying (the focus of Gremlin) and extend into deep learning, functionality that, to our knowledge, does not exist within any other graph database or graph computing system. However, the focus of this chapter is on functionality inherent to our SmartGraph design that strongly differentiates it from Gremlin and most graph computing systems. Namely, the SmartGraph is an asynchronous and concurrent system.

Asynchronous concurrent execution (as distinct from parallelism, as described in Section 4.1.2) is difficult in traditional distributed graph computing systems, such as Pregel, in part because the "think like a vertex" mantra (that was chosen to make "reasoning about programs easier") is typically implemented using Bulk Synchronous Parallel [148] methodologies. This involves top-level maintenance of lists of "active" vertices (and/or edges) during each "superstep" that all perform the same vertex (and/or edge) calculation. This can waste many computing cycles [149], since many graph algorithms do not converge at the same rate across different nodes. Proposed solutions to this problem often focus on maintaining sets of those nodes (or edges) that need to be updated during each iteration, and therefore, despite acknowledgment of asymmetric convergence rates, ultimately still use a synchronous iterative system.

The SmartGraph has a further novel feature that makes true asynchronicity a high priority; concurrent user requests and calculations. Consider, for instance, a distributed graph computing

implementation of two graph analytics which we wish to run concurrently; if one of the analytics is considerably faster than the other, the faster analytic will be held up by the global synchronization of the slower analytic. There are, of course, ways to address such issues (e.g. through multiple levels of synchronization), but it remains uncommon in graph computing systems to use concurrency and asynchronicity to solve multiple problems simultaneously (with asynchronicity instead employed to solve large instances of a single problem). The asynchronicity of the SmartGraph is instead designed to facilitate "multiple job asynchronicity" in order to serve the requirements of the many users who may be attempting to interact with the system at any given moment, in contrast to existing graph computing systems.

Despite its absence in existing methodologies, we show that concurrency is not incompatible with graph computing. Indeed, we demonstrate concurrency can dramatically increase the speed of graph computations. The business need for many simultaneous users existed in the graph database sphere, and thus, methods of solving this need (via concurrency) were developed. This business need has not traditionally been of such relevance for graph computing, yet we show that the techniques developed nonetheless are very effective when used for this purpose.

Thus the key difference between the existing methods for graph computing and the system outlined in this chapter is in having complex but manageable asynchronous concurrency of execution that eliminates the need for iterative supersteps and, more generally, the need for external macro-level management of the graph database and execution process. This chapter (and the system developed in this dissertation) facilitates this through subgraph level control, and by giving AI capabilities (through machine learning) to these subgraphs. Though there exist previous works that have used subgraphs as a base unit for organizing computations, such works either limit their asynchronicity to "within subgraph" computations [150], or define subgraphs as connected components [151, 152]. In general, we wish to define subgraphs with less restrictions than existing methodologies allow in order to take advantage of more local structure, and to give these local structures individual autonomy.

There are many clear benefits to having highly concurrent asynchronous routers capable of "thinking for themselves". For example, routers can receive, execute, and transmit graph query operations without needing to know (and limit their speed to) the global supersteps, as well as automatically handle locally relevant operations concurrently with other routers. In this chapter,

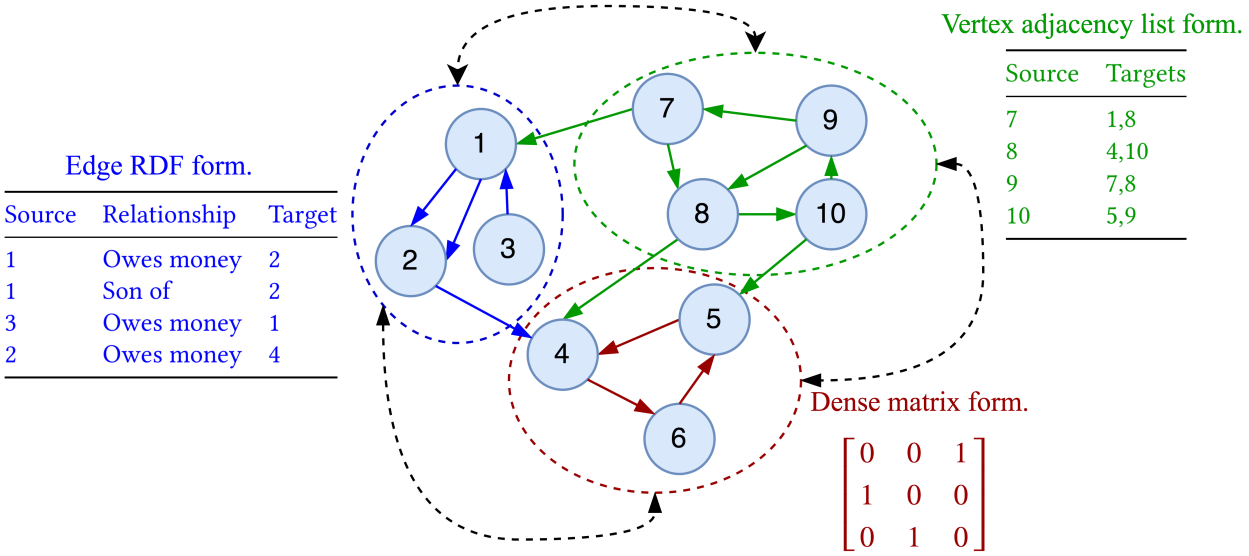


Figure 4.1: Toy graph with a possible goroutine router assignment.

we focus on outlining the design of the SmartGraph itself, and demonstrate how its asynchronous concurrency capabilities facilitate local AI. In particular, we demonstrate the value of this approach through the "router representation problem", where "routers" use machine learning methods and solve local optimization problems to great effect (i.e. well approximating the globally optimal solution).

4.1.2 Concurrency and Communicating Sequential Processes

In a modern graph database, it is a functional requirement not only that the system be able to deal with large amounts of data, but that the system be able to deal with a large amount of different requests with limited computational resources. The SmartGraph architecture facilitates massive concurrency for graph operations on graph-based data by explicitly tying this graph structure into a concurrency management mechanism.

Unlike parallelism which is typically thought of as simultaneous execution of very similar tasks, concurrent aware design can be thought of as a way of abstracting highly complex multi-part systems so that different processes (that may be related or entirely distinct) can interweave with one another (e.g. having overlapping lifetimes), resulting in the appearance of simultaneous execution. This appearance may be an illusion, such as a single-threaded CPU allowing a browser and word processor to run "at the same time" by quickly alternating cycles between each concurrent process

(and therefore the two tasks are not running truly simultaneously at the hardware level). It may also be a reality, in the case that process tasks are allocated to distinct threads of the CPU. Figure 4.2 outlines the differences (and similarities) between naive parallel execution and concurrent execution using three fictitious processes. Note that one can take advantage of the delay in Process 3 (e.g. representing some external servers' computation that needs to be complete before the process can be finished) using concurrency in both the single-threaded and dual-threaded scenarios. The dual-threaded and concurrent execution model takes advantage of parallelism more effectively than naive parallel execution of the processes; the ability to move process tasks between threads facilitates a consistent high load and shorter overall execution time.

Communicating Sequential Processes is a method of implementing concurrency in programs based on message passing. This work takes some liberty with the theory of CSP as a whole, and instead focuses on CSP as it is implemented in the programming language Go (also known as Golang) [153], developed by Google in 2009 with a focus on concurrency primitives as first-class citizens. Go is a language often described as a systems programming language, and is commonly used to design services, web-servers etc. It is with these capabilities in mind that we have created the SmartGraph in the Go language.

In Go, CSP is implemented through two main concurrency primitives: Goroutines and channels. A goroutine is essentially a very lightweight function that can be multiplexed onto different threads to be run concurrently with one another (and with the Main, which is itself a goroutine). Goroutines communicate not by sharing memory, but "share memory by communicating" through the use of the channels. In computer science terminology, this kind of simulated thread is called a "green thread" (as opposed to concurrency executed directly on the OS level thread) [154]. The simplest Golang channels "block" until there is a goroutine that wants to read from the channel at the same time as a goroutine that wants to write to the channel (at which point the channel permits the sending routine to push the message into the channel, and the reading routine to pull that message out).

Although goroutines (and green threads, in general) act like threads, they are not true hardware threads. This allows us to design a more lightweight concurrency mechanism, with many more goroutines than available threads. This is ultimately of critical importance, as it allows us to think

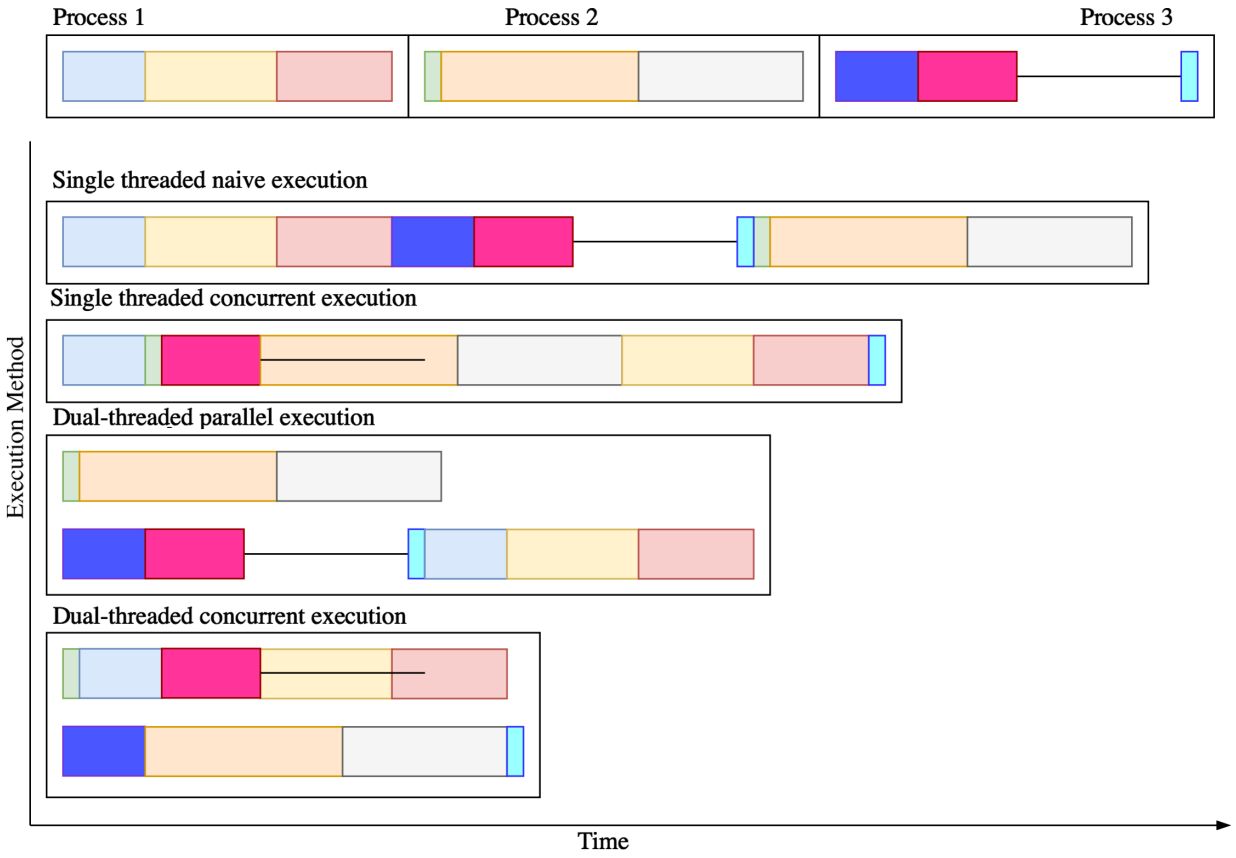


Figure 4.2: A comparison of concurrent and parallel execution in single and multi-threaded environments.

about concurrency at a higher level of abstraction, permitting us to simplify the management of graphs with a number of goroutine managed partitions at scales unheard of in the literature [155].

4.1.3 The Router

The SmartGraph concept is implemented in Go by explicitly tying goroutines to the graph structured data that is to be explored or used in algorithms to be executed. Individual goroutines define (not merely control) the subgraphs of interest, with the relevant graph structure, vertex and edge properties defined in the variable size stack corresponding to the particular goroutine. The goroutines are our method of implementing the routers, and the highly concurrent functionality of the goroutine is the mechanism through which we implement the communication and execution of analytics.

This approach contrasts strongly with the very strict definition of a subgraph (as connected components) used in prior works [151, 152], as it allows us to arbitrarily assign vertices and edges

to routers. Though this may seem similar to partitioning done at the network or thread level in distributed graph computing [155], recall that goroutines are far more numerous than threads, and balancing occurs by multiplexing goroutines onto threads, in contrast to finding a partition that provides a balanced cut of the graph to all available threads. This allows us to partition the graph into many more pieces than might be indicated by the number of OS level threads. Having separate graph structures in the routers facilitates taking advantage of local graph structure or optimizing with respect to locally requested operations. An example of this phenomenon, which we refer to as the "router representation problem" is explored in Section 4.3.

Through viewing this as the graph managing its own concurrent execution, one can do away with the top-level maintenance of sets of vertices, or algorithm iteration supersteps required by BSP methods. Concurrent execution is instead automatically handled by the router goroutines, as they will (through their status as a goroutine) immediately signal to the scheduler when they are ready (having received required data or messages from parent goroutines) to execute desired operations. This allows the system to be highly asynchronous, yet manageable thanks to the router abstraction. A basic example encapsulating a SmartGraph in goroutine routers is given in Figure 4.1. Observe the toy directed graph with no particularly noteworthy node or edge features. We show an example set of three goroutine routers that encapsulate the SmartGraph. Each router, defined by a dotted ellipse of a primary color, contains a subset of nodes and edges from the overall graph. The dotted black connections represent channels between the source and destination routers.

The network router is useful not only as an abstraction model for efficiently implementing and executing graph algorithms, it also inspires useful functionality for the SmartGraph. Network routers have the ability to maintain useful information in memory, such as routing tables. They also possess the ability to perform custom routing logic independently of other routers. There are both obvious and subtle ways in which a graph database and analytics platform aping this functionality provides benefits. For example, a straight-forward method of employing the benefits of local representation is in saving memory address space as highlighted by Figure 4.3. The "local" aspect of the router can also be taken further than simple indices. With routers able to act independently, we introduce the notion of "local subgraph representations", where the routers encapsulating a subgraph do so via different storage mechanisms and graph formats. For instance, one router may organize the subgraph it encapsulates using vertex adjacency lists, with another router learning

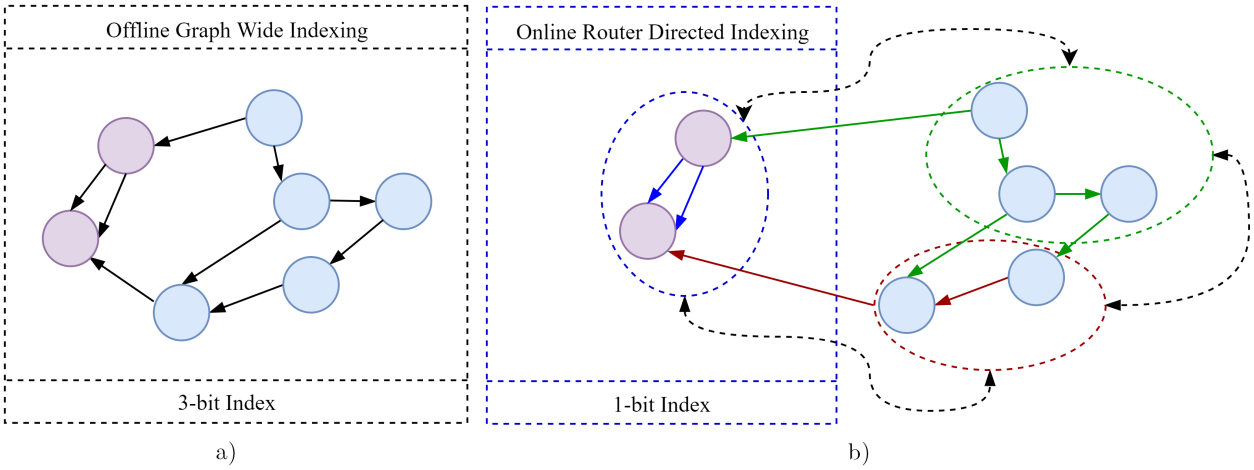


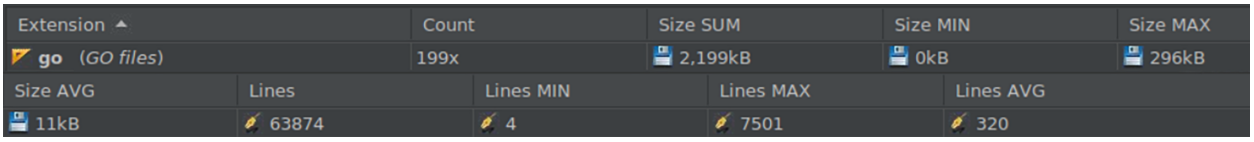
Figure 4.3: Toy comparison of graph-level and router-level vertex indexing.

to use an RDF framework. An example of this sort of router dependent representation is given in Figure 4.1. This approach has both computational benefits (since some operations have faster implementations in certain graph representations) and storage benefits (e.g. when using a sparse representation like CSR, or when compressing a subgraph consisting of nodes with high similarity).

4.2 System Implementation and Performance Metrics

As mentioned in Section 4.1.3, the SmartGraph system was designed and implemented in Google's Go programming language. In this section, we give a brief overview of the implementation. The overview is brief by necessity; as demonstrated by Figure 4.4, the project contains approximately 64k lines of code in nearly 200 files, and so giving a detailed account of all functionality in the system is infeasible. Instead, we focus on aspects of the implementation that we believe are particularly interesting and/or novel. This takes the form of Subsections below, or entire chapters where appropriate e.g. Chapter 3 and Chapter 5. In particular, material central to the system design and implementation is also discussed in Chapter 5 due to the central importance of deep learning functionality within the system. We also include performance metrics within this section in order to demonstrate the functionality of the system. These metrics were chosen specifically to demonstrate central theoretical and design concepts developed for the SmartGraph and this dissertation.

First we demonstrate in Figure 4.5 the macro-level logic behind the design of the SmartGraph. The reason we use this term is that this is how the system *appears* to act when observed at a high



Extension	Count	Size SUM	Size MIN	Size MAX
go (GO files)	199x	2,199kB	0kB	296kB

Size AVG	Lines	Lines MIN	Lines MAX	Lines AVG
11kB	63874	4	7501	320

Figure 4.4: SmartGraph code implementation size summary.

level of abstraction. However, this figure is clearly an approximation for a very simple reason; the SmartGraph is massively concurrent, and therefore its operation cannot be truly captured by a state diagram. At any given moment, the SmartGraph can have hundreds of thousands of different "states" (i.e. running goroutines) that are performing operations that are related or distinct, communicating or non-communicating, or originating from individual or multiple users.

4.2.1 Buffered Channel Management

A buffered channel is much like a regular channel (in that they are used to send information between concurrent goroutines), except that they have buffers. This means that it is possible to put information onto a channel as long as there is space available, and possible to read from a buffered channel as long as there is something on the buffer. This is in contrast to non-buffered channels, where messages only propagate along the channel when a read and write occur at the same time (i.e. one is blocked until the other occurs). These buffered channels form an important part of the SmartGraph router system.

When a router is initialized, a goroutine and a corresponding channel are created that together serve as an input to, and overall manager of, that router. When the router receives a request from "somewhere" (input by a user, hard-coded, created by a different router, or created *by that router itself* during a previous iteration) through the router channel (in our design, router channels are always considered inputs; for a router A to send a request to a different router B, it must send it through the channel corresponding to router B), the request is then placed on a buffer that is managed by the router goroutine. This buffer (paying a one-time memory penalty during construction of arbitrary size; all experiments conducted for this dissertation used a buffer for each router with a limit of 20000 requests to ensure we never reached full capacity even during very heavy workloads) is then processed in FIFO order by the central loop of each router object.

This central loop of the router goroutine determines the request type of the request read off of the buffer. Each request carries an information packet with the information relevant (and/or

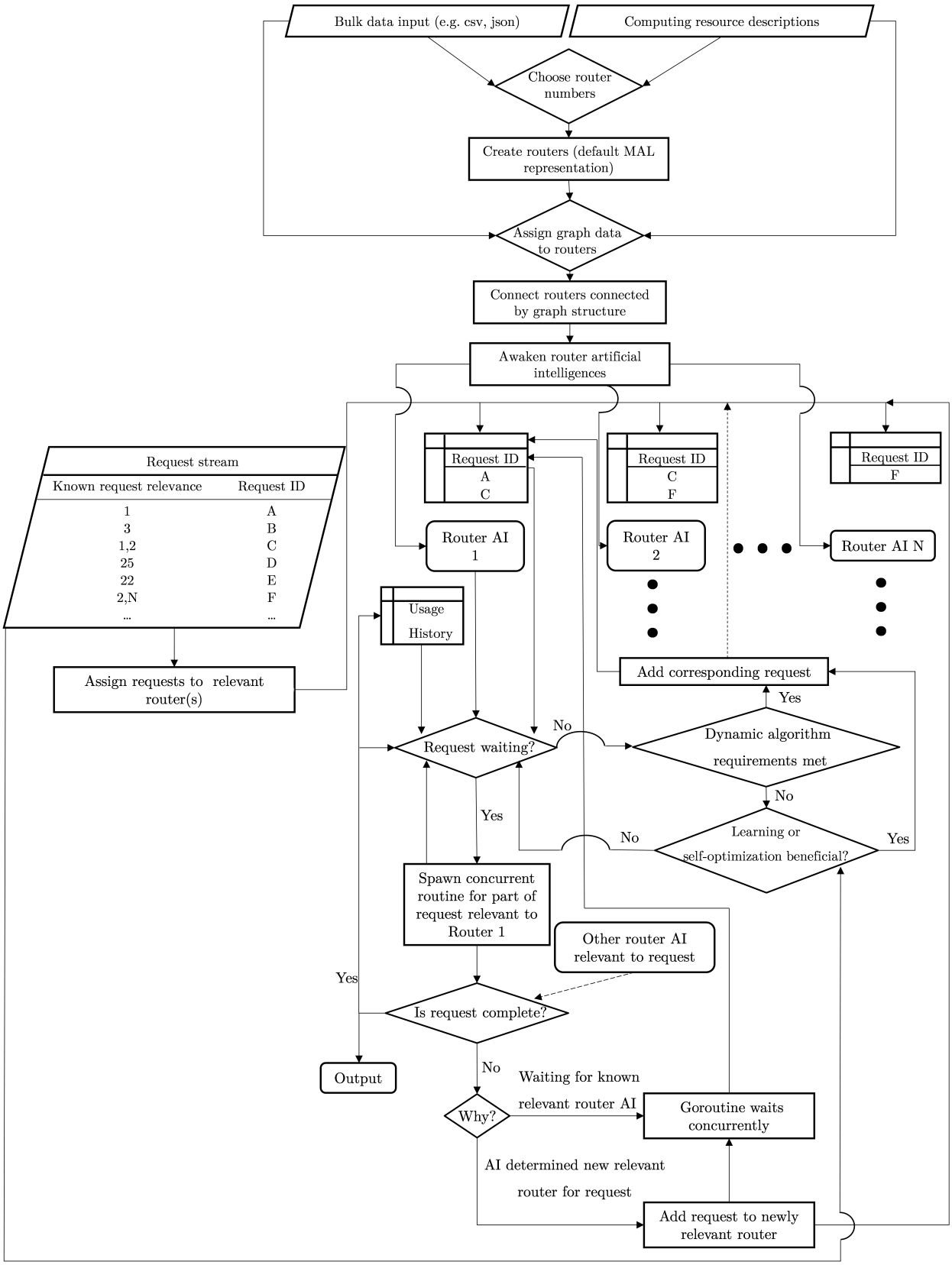


Figure 4.5: State diagram of the macro-level logic of the SmartGraph system.

required) for the corresponding request type. The central loop then spawns a new goroutine to execute the request and passes this information packet to the appropriate function (allowing the central loop to continue reading requests off of the buffer while the new goroutine processes the current request). Here we note that Figure 4.5 is simplified in that it has a separate "Request stream" and router level buffers. In reality, these systems are the same, and the requests stream directly in at the router buffer level (however, as indicated by the Figure, they are correctly targeted to the relevant router(s)).

There are a number of important categories of requests. The most basic of these are requests that do not require any sort of acknowledgment or response, which can be casually executed without concern. More complicated are requests that require a signal (such as verification that adding some new graph structure has been completed) or data (such as the result of executing an analytic). These signals or response data messages are the mechanism through which complicated algorithms consisting of many requests across many routers are executed.

This is accomplished through a special request type called a ContentRequest. A ContentRequest carries in its information packet another request, as well as a globally unique subchannel ID number (e.g. generated by a secure random number generator at the receipt of each message) and *a channel capable of returning the type resulting from the internal request*, which we refer to as the "return channel". Upon receiving a ContentRequest, the router stores into a map keyed by subchannel IDs the return channel, and then adds the internal request to its own buffer. This internal request is special, in that it includes the subchannel ID from the ContentRequest in its information packet. When this new internal request is executed by the router, the presence of the subchannel ID indicates to the router that the result of the internal request needs to be returned. The subchannel ID is used to key into the map of return channels, the appropriate channel found, and the result sent back upon it. While all of this is happening, the return channel may have been sent to another router B with instructions to read from that channel when possible and input the result along with a new request (i.e. a concurrent goroutine is created by router B to wait for the "return channel" to have something available; in this way, router B is free to continue processing available requests while it waits for the result of the original internal message to become available) into its own buffered channel of requests.

A simple example of this process is demonstrated by Figure 4.6. Router 1 has received two dependent requests, in addition to many other unrelated requests. The boundaries within each process box represent when a new operation in a process is ready to begin (and thus the only assumption made about the completion time of the previous operation is that it is completed by the time the next process division is given). The first request is to read the vertex weights of the neighbors of node β, and the second is to sum these weights. In this example, Router 1 creates a goroutine to read the neighbor weights of vertex β in Router 1 (Figure 4.6 is drawn so goroutine creation is drawn downward, and results are passed upward). While this is being executed, the created goroutines check and determine that vertex β exists in Routers 2 and 3 also, and therefore neighbors in these Routers must also be checked. The goroutine created by Router 1 thus creates requests as appropriate and sends them to the corresponding routers, which in turn generate new goroutines to execute the requests and return the results. While this is all going on, Router 1 is free to process the other requests sent to its buffer.

It should be noted that Figure 4.6 is a simplification in a number of ways. The most significant is that the SmartGraph system actually has a rule that routers only interact through requests, so when we "Return result on Channel B", we are actually sending a new request to Router 2 to send a request to Router 1 containing the result, which will then be combined with a similar process from Router 3 to allow the red goroutine to complete. This is part of the ContentReturn request type that we will not delve into beyond this description. As we can see, even "simple" programs become extraordinarily complex in their execution patterns in the SmartGraph system. Fortunately, the mechanics of how the Buffered Channel Management is set up mean that the user does not need to know any of this; it is all automatically managed by the interactions between the routers and the subchannel identification system as described above.

It is this system that facilitates the management of millions of concurrent operations with extraordinarily complex interdependencies. The SmartGraph also supports very complex requests: with requests in requests in requests, a single request on a single router generating multiple requests of different nature across multiple different routers of different type etc, but all of these more advanced requests are built on the foundation of the ContentRequest. Thus, we are able to execute complex OLAP analytics using a transactional system inspired by OLTP graph databases. Consider that it is only through thinking at the level of abstraction facilitated by green threads that this

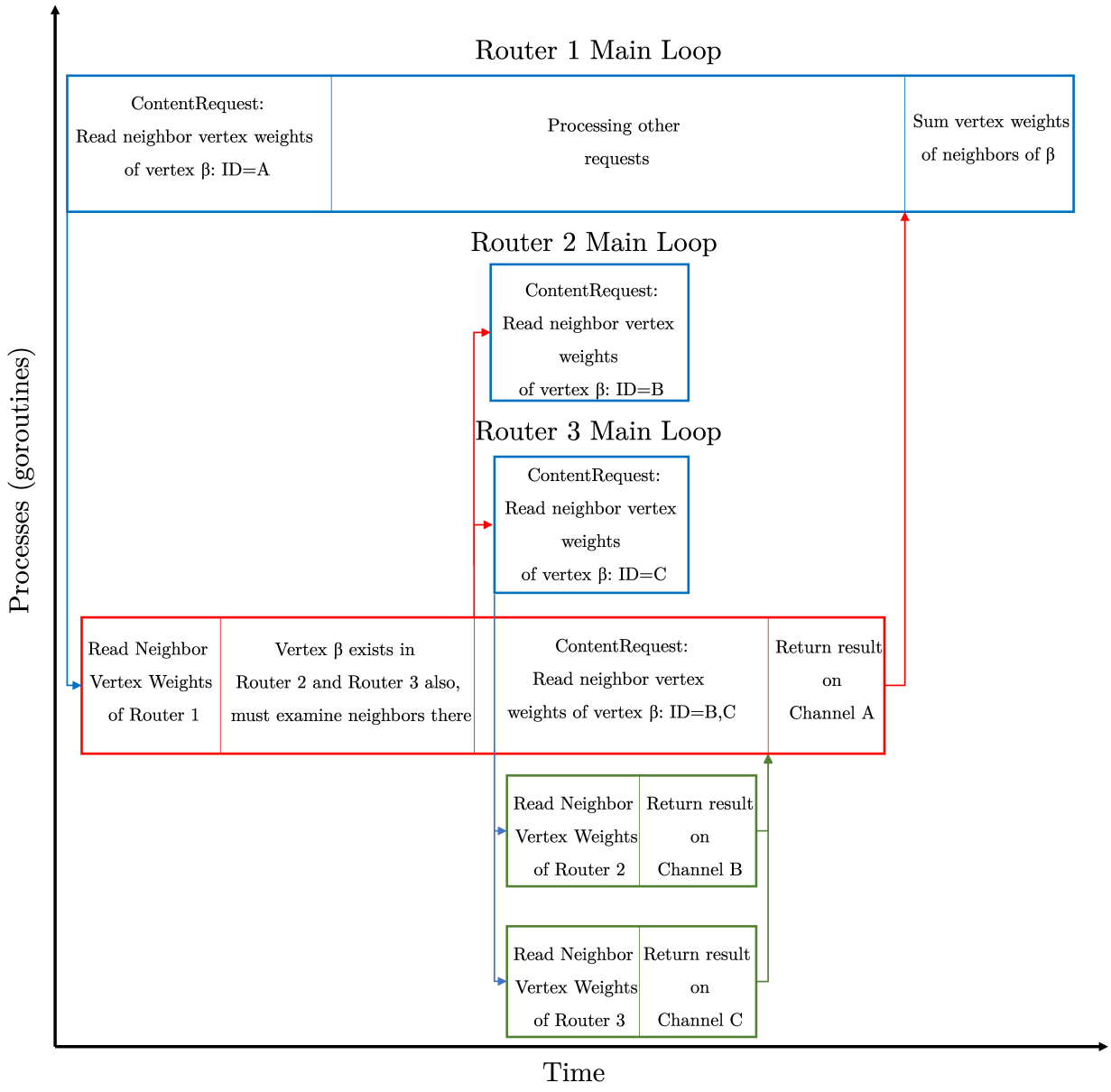


Figure 4.6: Concurrent execution of a ContentRequest and a dependent request.

system design is able to be conceptualized; the goroutines that manage each router routinely create new goroutines to execute each request, and essentially forget about them (unless a subsequent message requests the result of a prior ContentRequest, or the result is forcibly placed upon the buffer). In contrast, OS thread level concurrency management would have us manually running concurrent processes on a very small number of physical threads. It is clear from Figure 4.6 that such manual management would essentially be impossible. This ability to think larger in terms of the number of threads (virtual though they may be) facilitates not only an effective concurrent process management system, but also leads to tangible performance benefits, as we soon demonstrate.

4.2.2 Multiple Routers

Thus far, this methodology could technically be implemented with a single router. However, it should be clear it is not beneficial to do so. Though many requests are executed concurrently, each router has only a single loop extracting requests from the buffer, and only one request can interact with a given router's data at a time, otherwise we can have race conditions where different processes attempt to write to the same memory i.e. simultaneous property updates on the same vertex. The code in the SmartGraph is written to be completely free from race conditions, and deadlocks (where there is a cyclic dependency between goroutines waiting on data, meaning nothing is getting done and the program crashing) are naturally avoided by the Buffer Channel Management system. Consider Figure 4.6 once again; by virtue of having multiple routers, each of which have their own subgraph structure, we can read these structures concurrently, and thus execute analytics more quickly.

Some of the most tangible performance benefits of having multiple routers are demonstrated in Section 4.3, but we can observe them even at this stage. For example, consider Figure 4.7, that demonstrates execution time on our machine (an 8-core 16 thread Intel(R) Core(TM) i7-7700 CPU @ 3.60GHz with 64GB of 2400MHz RAM) for a task involving the addition of 10000 property vertices (note that this is a task that interacts with only individual routers, since it is a request that does not require router interaction) across SmartGraph setups with different numbers of routers. Observe that increasing the number of routers readily reduces the execution time of the problem. Furthermore, we get benefits with more than 16 routers; that is, it can be beneficial to have more routers executing analytics than the number of threads present on the machine. Eventually, we

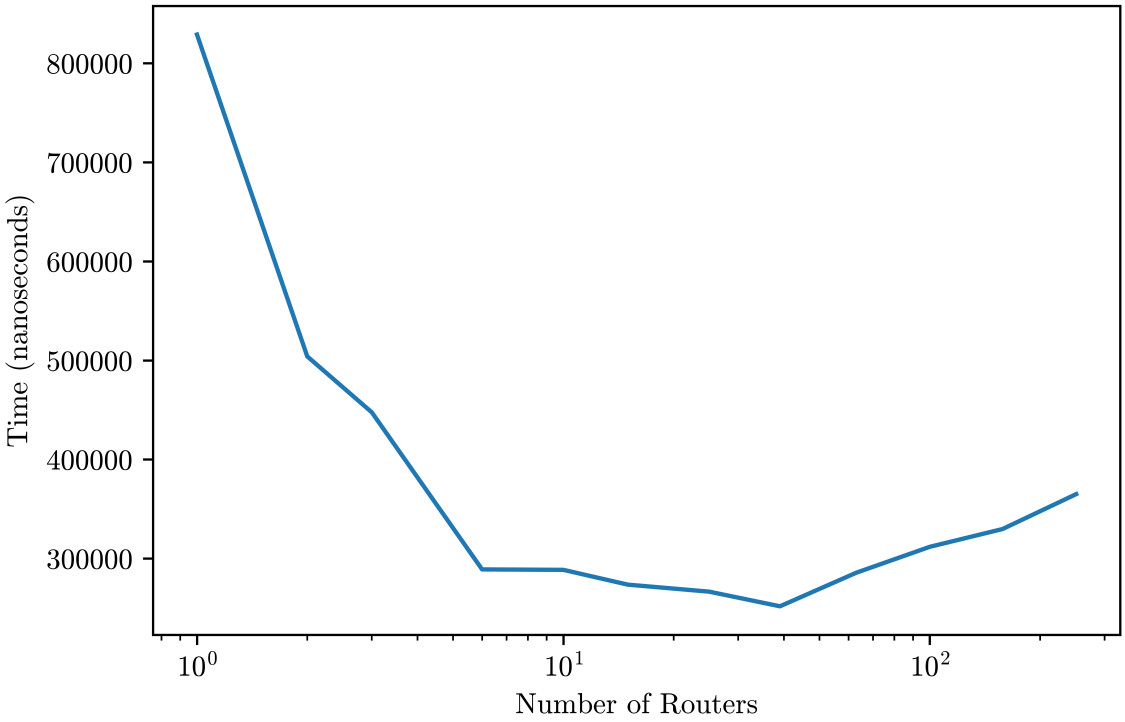


Figure 4.7: Increasing the number of routers for a vertex input task.

get to a point where the overhead cost of maintaining multiple router systems is greater than the concurrency benefits of additional routers, and the execution time begins to increase once more. We note that the appropriate number of routers is problem dependent. In general, communication within a router is faster than communication between routers, but having an increased number of routers is beneficial when requests involve different parts of the graph (e.g. as would be the case for a batch of requests to analyze transaction networks of many individuals within a financial transaction network). The key here is that having a single router is almost never optimal, nor is the other extreme of having an independent router for every vertex in the graph.

4.2.3 Graph Representations

A key functional capability of the SmartGraph is the ability to represent graph structured data using multiple representations. This functionality combines with multiple routers to be beneficial for both efficiency and memory usage. We briefly outline some representations supported by the

```
type RDFSubGraph struct {
   sync.RWMutex // ReadWrite mutex to restrict read/write or write/write clashes.
   EdgeList *sortedmap.SortedMap // Sorted by destination vertex id.
   VertexList map[int]Vertex // Keyed by map ID to object of general Vertex type.
   RouterId int // Associated RouterId
}
type MapAdjacencyListSubGraph struct {
   sync.RWMutex // ReadWrite mutex to restrict read/write or write/write clashes.
   VertexMap map[int]*ALVertex // Pointer to ALVertex type that contains edges.
   RouterId  int // Associated RouterId
}
type ListAdjacencyListSubGraph struct {
   sync.RWMutex // ReadWrite mutex to restrict read/write or write/write clashes.
   VertexList []ALVertex // Locally stored list of vertices that contain edges.
}
type CscSubGraph struct {
   sync.RWMutex // ReadWrite mutex to restrict read/write or write/write clashes.
   Indices []int // Used to find edges in the CSC representation.
   Indptr  []int // Used to find edges in the CSC representation.
   AdjacencyEdge           []Edge // List of edges ordered by INTERNAL index.
   VertexMap              map[int]Vertex // VertexMap is keyed by the EXTERNAL index.
   InternalToExternalIndex map[int]int // Map to move from internal to external index.
   ExternalToInternalIndex map[int]int // Map to move from internal to external index.
}
```

Figure 4.8: Graph representation data structures as implemented in the SmartGraph.

SmartGraph, and demonstrate how different operations are often suited to particular representations.

The most important representations supported by the SmartGraph are the Matrix Adjacency List (MAL), and the Resource Description Framework (RDF) [156], as these are the representations used to demonstrate the Router Representation problem in Section 4.3 below. These representations are implementations of the standard adjacency list and edge list forms. In an adjacency list, each vertex contains a list of outbound edges (and thus the MAL representation performs particularly well when executing an ordered traversal of all outbound edges, as demonstrated by Figure 4.10), and so space is not wasted on non-existed edges as in an adjacency matrix form. For an edge list, separate lists of vertices and edges are maintained (the lack of complex property structure gives RDF an advantage when performing simple tasks that don't require lookup, such as batch-loading vertices; demonstrated by Figure 4.9). In addition to the MAL representation of the adjacency list, the SmartGraph also contains the List of Adjacency Lists (LAL) representation (where vertices are stored into a list rather than a map that can be keyed into), inspired by the "Index Free Adjacency" storage method of Neo4j [138]. Due to their frequent use in tensor computations, we also include a Compressed Sparse Column (CSC) representation. A property of the CSC representation of a sparse matrix is that column slicing is extremely efficient. Since columns of an adjacency matrix refer to

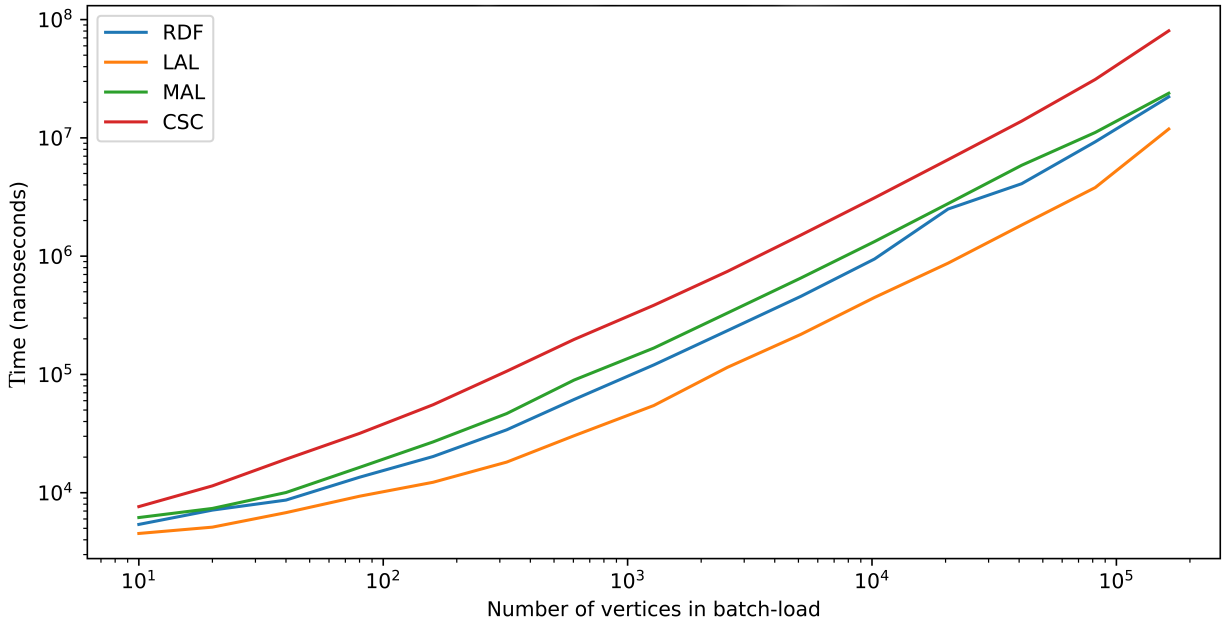


Figure 4.9: Execution time comparison for different representations performing a property vertex batch-load operation.

inbound edges, orderly traversal of inbound edges is thus extremely fast in the CSC representation, as shown by Figure 4.11.

The code describing these data structures is given in Figure 4.8. An important point to note here is that the RDF representation has edges ordered by destination vertex ID such that there is an increased lookup speed for inward edge lookups and traversals (as a contrast to the outbound edge focused MAL representation). This is why RDF begins to outperform representations other than CSC in Figure 4.11. The benefit requires a large graph due to the overhead inherent in having an ordered map structure. Also observe that the adjacency list representations use a special kind of vertex structure (that satisfies the general Vertex interface used in the other representations) that contains edge objects (in their entirety) in addition to the regular vertex information (such as vertex ID, type, vertex properties etc), which is why they tend to be slower for small operations that don't require much inspection of the graph.

It should be clear from these results how and why different representations are suited to different kinds of operations (and vice versa). We explore this problem in detail in Section 4.3.

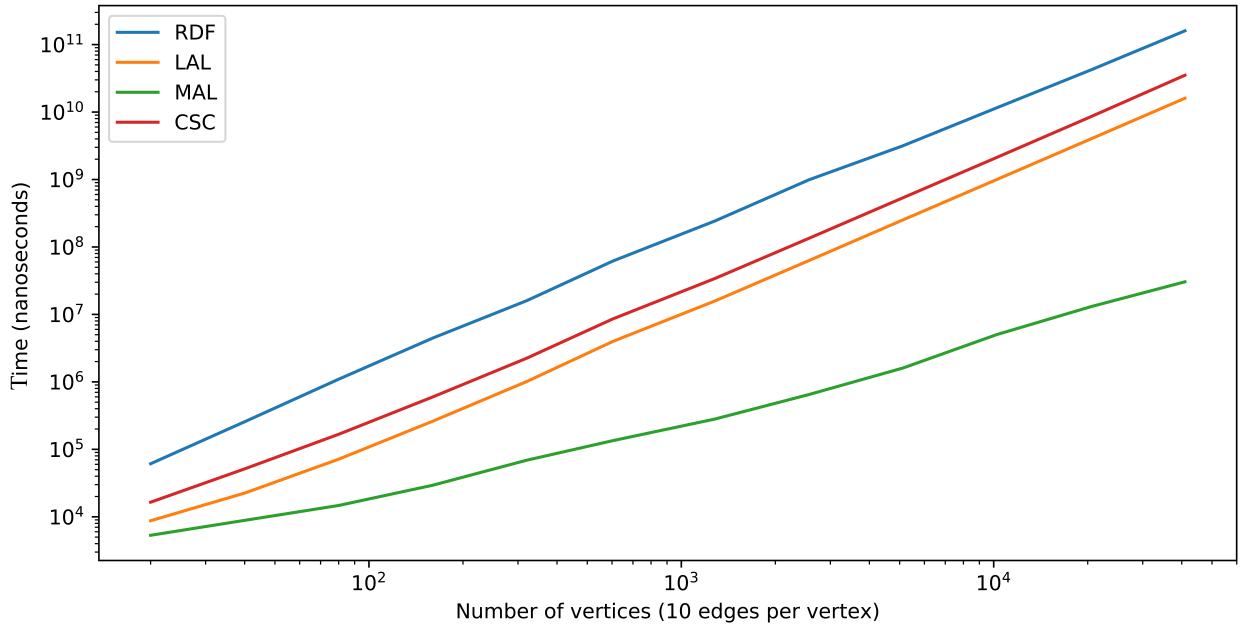


Figure 4.10: Execution time comparison for different representations performing an outbound traversal of all edges in the graph, vertex by vertex.

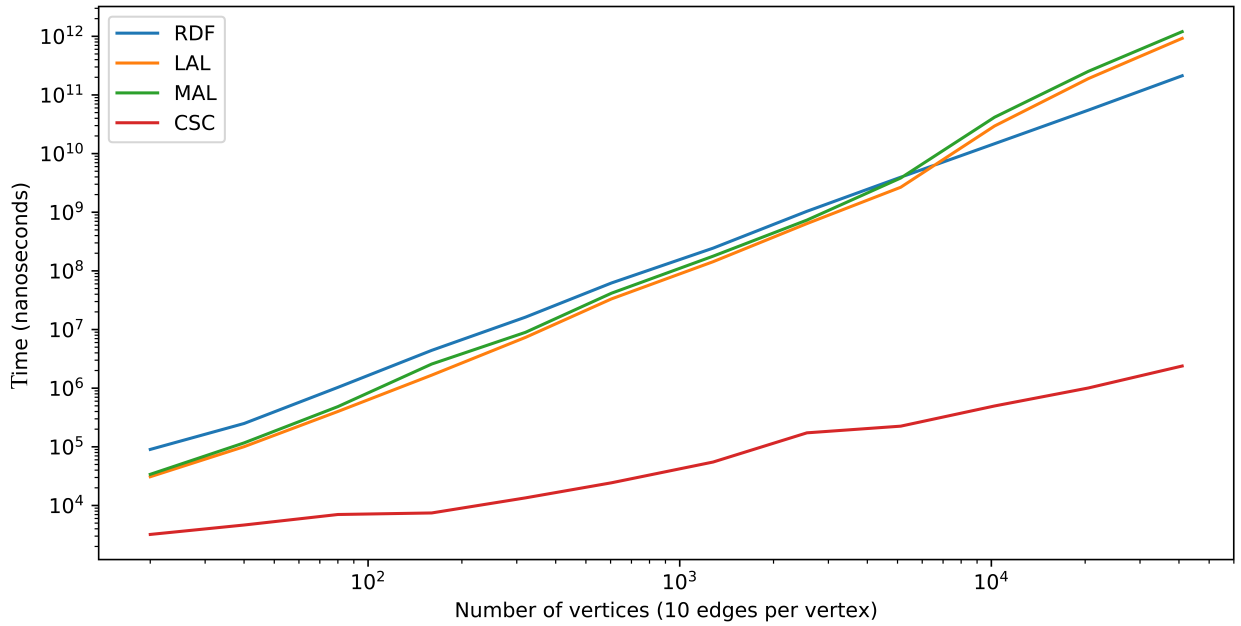


Figure 4.11: Execution time comparison for different representations performing an inbound traversal of all edges in the graph, vertex by vertex.

4.2.4 Total Asynchronicity Control

In prior sections, we have discussed how the use of multiple routers and the Buffered Channel Management system allow the SmartGraph to achieve extraordinarily high levels of asynchronous execution, without management becoming overly complicated due to the automatic management of this asynchronous execution. Complexity is part of the reason that BSP methods are so popular. The other reason is that BSP allows for similar operations to be batched together, and executed at the same time (gaining performance benefits from economies of scale). Note however that while this is a situation that will occur regularly in a strict graph computing setting (where we are focusing on single large analytics at a time), it is not expected to be particularly common in a graph database setting (where the different requirements of many simultaneous users can make such batching potentially difficult with less straight-forward benefits).

Nonetheless, the ability to batch operations can be beneficial under some circumstances. For this reason, the SmartGraph system does have capability for requests to be batched, and this is achieved in a very natural way. To batch requests together, we simply assign them the same subchannel ID (though these IDs are generated as unique values, the unique values can then be assigned to multiple requests if desired). If a router receives a request with an ID that already exists within the system (recall that subchannel IDs of requests are stored within a map, and thus this check is an extremely quick O(1) operation), it automatically creates a new goroutine that keeps track of all matching requests, this goroutine will then only allow a given request in the batch to return (via the creation of ContentReturn requests) any information or results when *all* requests in the batch are complete.

This simple feature means that the SmartGraph gives users an essentially arbitrary level control over asynchronicity. Unlike BSP methods, and very regimented computation methodologies like the MapReduce framework [157], *any* combination of requests can be manually batched together to achieve some goal (e.g. speed-up through cache-access of the same data). This process is always manual since deadlocks are possible if cyclically dependent requests are batched together (it is the responsibility of the user to ensure that this does not occur). If manual batching is not specified (or for any non-batched requests when batched requsts are also present), the SmartGraph will continue to execute requests with the approach to asynchronicity determined by the Buffered Channel Management system.

4.2.5 Shadow Vertices and Replication

By virtue of having multiple routers that each contain a subgraph of the overall graph structure, we are inherently partitioning the graph into multiple pieces. Since graph structure is interconnected (across partitions), this by necessity means that the SmartGraph requires functionality to allow connected parts of the data to be recognized at connected, even across routers. The most "natural" way of envisioning this is that "edges cross between routers". This "edge cut" method is therefore what the user of the SmartGraph *sees* when working with the system. However, much research [155, 158] has been completed that indicates a somewhat counterintuitive result; it is typically better for both communication and storage overhead for *vertices to cross routers*.

In practice, this means that vertices are duplicated across routers, and each router maintains a map of its vertices that are duplicated elsewhere, and the duplicated locations. We refer to duplicated vertices as "shadow vertices". A serious problem in many distributed graph computing systems is that these vertex copies can become out of sync (e.g. if a machine updates a vertex at the same time another machine is using it in a computation). There are many approaches for this problem, such as informing any duplicate machines to halt and redo any calculations that used an old value of a vertex property etc. However, it was determined that this would be too costly in the SmartGraph system where we can have many more partitions than is traditionally considered. We therefore use a "best effort" approach. Whenever a vertex on a given router, that has duplicates, is changed, requests are automatically generated by the router and sent to the routers containing the duplicates with an instruction to update the content to the most recent vertex copy. This means that if a router receives a vertex update request for a vertex that it has updated more recently, the request is ignored, and it is the more recent update that will be propagated (this prevents an infinite loop of update requests where a vertex updates in response to an update request from another router, and generates an update request as a result).

This approach is usually sufficient, but can mean that a change may be forgotten by the graph. In many cases, such as PageRank, this is not necessarily a problem, as the next iteration may synchronize the vertices in time for the subsequent iteration. In cases where it is critical, a user may use the batching methodology outlined in Section 4.2.4 above to ensure that any dependent calculations do not occur until all vertex copies have been updated.

As we mentioned, this complexity is abstracted away from the user, who instead sees edges crossing routers. Thus when a user enters a request for an edge to be created that crosses the boundaries of routers, what actually occurs is that a request is sent from the source router to the destination router for a copy of the vertex. This vertex is then added to the source and destination routers' list of duplicated vertices, and the edge is stored on the source router from the source vertex to the duplicated "shadow vertex" that was just created.

Thus, any request regarding vertices that are duplicated elsewhere (recall this information is stored in a map, so this check is a quick O(1) operation for each vertex, regardless of representation) can quickly be resolved by the SmartGraph system. In Figure 4.12, we demonstrate the execution time of a set of requests to add edges to an existing partioning of vertices across routers in a SmartGraph system. The destination routers (whether internal or "external") of the edges are chosen according to the x-axis. We observe that as we increase the proportion of "external" edges from 0 to 1, the execution time increases accordingly. However, this increase is not too large, staying within two orders of magnitude for most representations. Note that the case of a large proportion of edges crossing routers is an extreme and unusual one chosen to demonstrate this performance characteristic, as most graph partitioning algorithms explicitly partition in order to minimize either the number of external edges, or amount of communication between shards of the partition [155].

4.3 The Router Representation Problem

The representation problem, as defined here, is closely related to the more generic problem of selecting data structures either at compile- or run-time, in order to minimize memory usage, execution time, or some combination of the two. In contrast to this work, the literature focuses almost exclusively on optimization with respect to standard container objects (such as lists, sets, arrays, hashMaps etc) [159–161] rather than graph specific data structures. A few papers do address the problem in a more graph specific context; for example, [162] directly considers the impact of basic graph operations and representations on execution time. However, it only uses the vertex adjacency list form and investigates how the single graph representation can be constructed by different container types, and so does not directly consider multiple graph specific data structures

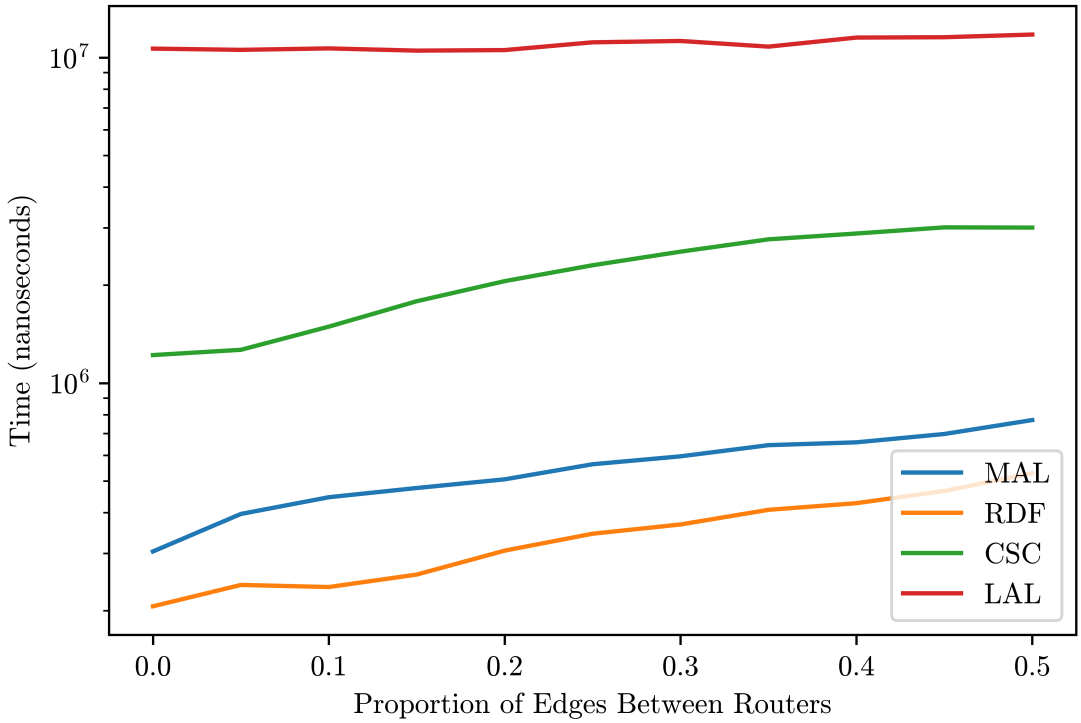


Figure 4.12: Performance of different representations for an edge addition task as the proportion of edges crossing router boundaries increases.

(i.e. alternatives to the adjacency list). Furthermore, [162] considers representations that "change" over time, but at any given time applies a single data structure to the entire graph. This means that the method cannot benefit from local graph structure. In contrast, [163] explicitly investigates different graph structure representations (both the adjacency list and the adjacency matrix), but again only considers their choice in application to the entire graph.

The data structure selection problem has been approached from three primary directions. The first is that of optimization, where benchmarking is used to create a function that approximates the execution time of a series of operations [162]. The second is using machine learning to generate rule sets that can be used to determine the choices of representations (e.g. "if BFS is called on a graph with density greater than 25 percent, use an adjacency matrix, else use an adjacency list") [159]. The final commonly used method is to simply provide a framework for implementing swap rules, and allow the user to specify precisely what those rules are manually [160].

Unique to this work, we explore methods for choosing local graph representations (that is, structural representations that apply locally rather than to the whole graph) by solving a set of optimization problems over learned models that consider both the local graph data, as well as the graph operations requested in relation to that data. Furthermore, we do so by deconstructing the problem such that a problem that initially appears exponential in complexity becomes linear in the number of routers by the number of router representations (and that can therefore be solved concurrently by the routers). In Section 4.3.1 we introduce the basics of the representation problem. In Section 4.3.2, we establish the basis of solving the representation problem under idealized circumstances, and in Section 4.3.3, outline how we learn the functions required to solve the problem in practice. This involves using average router behavior as an approximation, which we argue in this section and demonstrate in Section 4.3.4, is an approximation that ultimately works very well.

4.3.1 Representation Methodology

Let $G = (V, E)$ be a known property graph, i.e. a graph with properties attached to vertices and edges, where we allow multi-edges (e.g. representing different types of relationships between the same two objects). Let $P = (P_1, P_2, P_3, \ldots, P_k)$ denote a partition of the edge set E. Let $V_i = \{u : (u, v) \text{ or } (v, u) \in P_i\}$. We allow for a vertex v to be an element of more than one V_i (thus we are in the "vertex cut" framing as actually used in the SmartGraph; see Section 4.2.5 for more details). In a SmartGraph database, a router $\mathcal{R}_i$ encapsulates the subgraph (V_i, P_i).

We emphasize that the partitioning introduced here should not necessarily be considered in the same vein as traditional graph partitioning [155], as we are operating on a single machine (though with many cores) unconstrained by threads (due to the nature of green threads), and as a result we are not overly concerned with partition balance.

Let S denote the set of all graph representations allowed by the SmartGraph. Let $R_i \in S$ denote the representation employed by router $\mathcal{R}_i$ for (V_i, P_i), $i = 1, \ldots, k$, with R the vector of all such representation choices. In this work, the allowed set S of representations includes the adjacency list (wherein vertices are stored in a hash-table like structure, with lists of outbound vertices), and the Resource Description Format (RDF), which stores separate tables of edges and vertices (we deliberately use the term tables here, as the use of RDF can be thought of as functionally similar

to implementing a graph database in a traditional relational database system). See Section 4.2.3 for more details. These two representations are by far the most common method of representing graphs in practice. For example, the RDF format is used in Spark, and it's GraphX [158] and GraphFrames [164] packages, whereas adjacency lists are used by Neo4j [165], a leading commercial graph database system.

There are many more graph representations, including adjacency matrices (often impractical in practice due to large storage requirements), sparse matrix representations (wherein a user is more focused on efficiently performing mathematical operations than graph operations; though the two are often closely connected), and representations with particular properties (such as allowing directed graphs only, disallowing multi-edges, cyles) etc. There are also custom representations designed to work efficiently with GPU operations [166, 167]. Thus it should be noted that the combination of representations used in this chapter (and solved by the SmartGraph router abstraction herein) is not intended to argue that it will always result in the most efficient representative structure. Instead, we are arguing for a methodology for combining arbitrary different representation possibilities, and using the RDF and adjacency list as our representations for purposes of demonstration.

4.3.2 Predicting Execution Times and Choosing Representations

We represent a database job $\mathbf{x} \equiv \{x_1, x_2, \ldots, x_M\}$ as a sequence of M known operations, that we classify as either "complex", or "basic". Complex operations are those that internally execute other operations (either complex or basic) to complete their execution, those that require knowledge of multiple routers (such as inspecting the number of vertices in a given router that are duplicated in other routers), and those that requiring complex input. What remains we describe as basic operations. Each operation $x_a \in \mathbf{x}$ is executed on some graph structure of some size (e.g. a number of vertices, or a number of edges). The size of this structure may or may not be known in advance for each operation (consider a job of two hop neighbor traversals, where the number of neighbors traversed in the second operation depends upon how many neighbors were traversed in the first operation), and we thus introduce the term $|x_a| \in \mathbb{N}$ to represent the size of the structure upon which x_a will execute.

We allow for directed acyclic dependence between operations of a job, i.e. all parents of an operation x_a, $\pi(x_a)$ must be executed before x_a. Let $\mathcal{B}$ denote the set of l basic operations. We assume that the average run time for a basic operation $b \in \mathcal{B}$ on a graph with n vertices and m edges, and representation $r \in S$, is given by a function $h(b, m, n, r)$ (which for the moment we assume is given; Section 4.3.3 will outline how we learn this and other necessary functions).

Furthermore, we define a function $\mathbf{h}(\sigma, m, n, r) = \sum_{u=1}^{l} \sigma_u h(b_u, m, n, r)$, where $\sigma \in \mathbb{N}^l$ is a vector of basic operation counts (such that the element σ_u corresponds to the number of executions of basic operation b_u), and $\mathbf{h}$ is the time to execute σ counts of the basic operations without concurrent execution. Note that an operation initiated on router $\mathcal{R}_i$ may have to initiate calls to other routers in order to complete the operation. For example, since vertices are possibly duplicated, a "traverse neighbors" operation may require calls to other routers to check if those vertices exist there too, and get neighbors of such duplicates also. Let $c_{ij}(x_a, |x_a|) \in \mathbb{N}^l$ denote the number of calls of basic operations $\mathcal{B}$ initiated on router $\mathcal{R}_j$ when the operation is initiated on router $\mathcal{R}_i$. The non-concurrent execution time $T(x_a, |x_a|, i, R)$ for an operation x_a initiated on router $\mathcal{R}_i$ is then given by

$$T(x_a, |x_a|, i, R) = \mathbf{h}\big(c_{ii}(x_a, |x_a|), |V_i|, |P_i|, R_i\big) + \sum_{j \neq i} \mathbf{h}\big(c_{ij}(x_a, |x_a|), |V_j|, |P_j|, R_j\big). \tag{4.1}$$

However, in practice jobs are sequences of operations, where the number of relevant graph objects for a job can depend upon its parents. For example, consider a job $\mathbf{p} := \{x_1, x_2\}$ with $\pi(x_2) = x_1$ on a directed graph G (weighted by w) with partition P, where x_a is an operation "traverse edges from current vertex set if $w \geq \bar{w}$". For x_1, we assume that we have some initial vertex $z \in V_i$ as the current vertex set. However, we do not know in advance the size of the vertex set to which x_2 will apply. This means that in addition to knowing how many basic operations (and their type) σ are required to execute a given operation, we also need a count of how many graph objects (e.g. vertices or edges) are filtered through by parent operations. We define $f_j(x_a) \in \mathbb{N}$ as a function that takes an operation and returns a count of the graph objects within partition part j that survived the filtering of the parent operations $\pi(x_a)$. Then the non-concurrent execution time of job $\mathbf{p}$ is given by

$$T\left(\mathbf{p},i,R\right)=T\left(x_1,|x_1|,i,R\right)+\sum_j T\left(x_2,f_j\left(x_2\right),j,R\right), \tag{4.2}$$

where the first term corresponds to execution x_1 on the initial router, and the sum represents running the subsequent operation on the "filtered" nodes. We have overloaded the notation T to correspond to runtimes for either jobs or operations, with the meaning clear from context. Note that $c_{ij}(x_a,1)\cdot|x_a|\neq c_{ij}(x_a,|x_a|)$ because there may be duplicates (e.g. multiple edges pointing to the same vertex) that could otherwise cause a vast overestimation (e.g. consider a fully connected graph) in the number of objects for the next operation, and therefore a large error in time estimation.

It is easy to see how this approach extends to a much more general job $\mathbf{x}$ as

$$T\left(\mathbf{x},i,R\right)=\sum_{x_a\in\mathbf{x}}\sum_j T\left(x_a,f_j\left(x_a\right),j,R\right), \tag{4.3}$$

where we assume $f_i(x_1)$ is a count some known set, and $f_j(x_1)=\phi\,\forall i\neq j$. With this execution time function, we therefore wish to solve the following optimization problem:

$$\min_{\{R=(R_1,\ldots,R_k):R_i\in S\}} T\left(\mathbf{x},i,R\right)=\min_{\{R=(R_1,\ldots,R_k):R_i\in S\}}\sum_{x_a\in\mathbf{x}}\sum_j T\left(x_a,f_j(x_a),j,R\right), \tag{4.4}$$

Problem (4.4) is a very complex combinatorial problem (as there are exponentially many, $k^{|S|}$, possible choices of the router representation vector R), where the representation of any given router i can influence the performance of operations sent to different routers j. Thus, the choice of router representations must be made globally, with no opportunity for parallelization. However, we observe that we have an important opportunity, by virtue of having established total counts of basic operations and their router occurrence locations.

Consider that the number of routers, and the length of the job $\mathbf{x}$, is finite. We can therefore interchange the order of summation on the RHS. Also, each complex operation ultimately results in a set of basic operations distributed across the routers, and that *we know this distribution of basic operations before solving the optimization problem* because it is not dependent upon the representation, rather only on how complex and basic operations are defined. This means that,

supplied with appropriate c, f functions, we can rewrite the job $\mathbf{x}$ as a job $\mathbf{y}$, such that $\mathbf{y}$ consists only of basic operations (i.e. executing $\mathbf{y}$ executes $\mathbf{x}$). Furthermore, we can group the basic operations by router, such that for $\mathbf{y} = \{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_k\}$, $\mathbf{y}_i$ is the set of all basic operations for router $\mathcal{R}_i$. That is, we have the following very significant result:

$$\underset{\{R=(R_1,\ldots,R_k):R_i\in S\}}{\arg\min} \sum_{x_a\in\mathbf{x}} \sum_j T\left(x_a, f_j\left(x_a\right), j, R\right) = \left[\underset{R_i\in S}{\arg\min}\, \boldsymbol{h}\left(\boldsymbol{y}_i, |V_i|, |P_i|, R_i\right)\right]_{i=1,\ldots,k}, \tag{4.5}$$

where we are abusing notation in assuming $\mathbf{y}_i$ also has an appropriate vector form of counts of basic operations on $\mathcal{R}_i$. There are two key points to (4.4). The first is that we have successfully decomposed a very complex interdependent problem with exponentially many possible solutions, into a linear number of problems in the number of routers (represented as a vector of problems on the RHS) with a linear number of solutions (in the number of representations), resulting in a vast reduction in problem complexity. Furthermore, these optimization problems are local to each partition part. This means that each router can locally solve the simple problem associated with itself (potentially concurrently), and these locally optimal solutions together combine to form the *globally* optimal solution of Problem (4.5).

One aspect of this representation problem is that the problem as designed was explicitly constructed to avoid the use of concurrency (i.e. all runtime formulations and approximations assume there is no concurrency within the system). This was necessary in order to develop a feasible formulation (for as we noted earlier in this chapter, the mechanisms for concurrent execution are extremely complex). As such, the cost function in (4.5) is, strictly speaking, only a surrogate for what we truly seek to minimize; the runtime of a set of operations *with* concurrency enabled (this is why the problems outlined herein are framed as arg min). This formulation is therefore a surrogate based optimization approach [168, 169]. Yet, as we observe in Section 4.3.4, using this non-concurrent placeholder works extremely well in practice (where all experiments in Section 4.3.4 have concurrency enabled, as part of the SmartGraph system upon which all experiments were run). Though the addition of concurrency considerably changes how operations are executed within the system, models of how operations would be executed with concurrency disabled provides a powerful surrogate.

4.3.3 Learning Filtering, Counting, and Timing

In practice, we do not know the functions c_{ij}, f_j, and h unless we first run the operations, which is contrary to our intent of choosing representations before execution. Instead, we seek to learn approximations $\bar{c}_{ij}$, $\bar{f}_j$, and $\bar{h}$. To learn the approximations $\bar{c}_{ij}$, $\bar{f}_j$, we send sample operations to *each router* and learn based upon the average responses of each router. In job $\mathbf{p}$, different routers may have different vertex degree (affecting basic operation counts and $\bar{c}_{ij}$ because edges must be checked, even if they do not satisfy the condition to be passed to the next operation), and different edge weight distributions (affecting $\bar{f}_j$ and the number of objects passed to the next operation). In contrast, we learn $\bar{h}$, our basic operation runtimes, as a global precomputation for each representation type.

We wish to have a range of operations that is sufficiently flexible such that they can be composed into more complex graph operations. This means that their combination should be sufficiently expressive. In the extensive literature on graph query languages [170–172], the expressiveness of query formulations has been a central area of research, with expressiveness often traded off against complexity and efficiency. In practice, industrial graph query languages like Cypher and Gremlin have not been theoretically analyzed to any significant degree due to the complexity facilitated by the wide range of possible query operations permitted [173].

Query methodologies typically fall into two categories; path queries [170], and pattern matching [174]. For purposes of demonstration within this chapter, we choose to use a path query approach rather than a pattern matching approach. We choose this approach because it is clearly suitable for the methodology outlined in Section 4.3.2 (and because this is the approach used in Chapter 5 to facilitate new deep learning capabilities on graphs, due to the similarity with the execution of computational graphs). Consider that a path query typically has a form $I \overset{cond.}{\rightarrow} J$ where we are seeking paths from graph object set I to (potentially unknown) graph object set J that must satisfy conditions *cond.* along the path. There is an extensive literature on graph query algebras, and we refer the reader to [171] for a recent survey on the topic. For our purposes, we define path queries by chaining single hops with conditions on edges and/or vertices. Observe that this maps very neatly into the functions c_{ij}, f_j, and h. We have some path query that is decomposed into a series of stages, and conditions upon those stages. Each stage requires observing a certain number of graph objects (related to c_{ij}), only some of which survive to the next stage (related to f_j), and

Table 4.1: Examples of basic and complex operations

Basic Operation	**Complex Operation**
AddInternalEdge	AddExternalEdge
AddVertex	GetSharedVertices
GetVertex	ExploreNeighborsOutsideRouter
GetInternalEdge	FilterEdges
CheckVertexExists	FilterVertices
CheckInternalEdgeExists	CheckRoutersConnected
UpdateInternalEdgeProperties	External ContentRequest
UpdateVertexProperties	ComputationalGraphQuery
ExploreNeighborsInRouter	ExecuteOptimization

these are internally composed of basic graph operations (related to h, see Table 4.1) with timing dependent upon representation.

The existence of properties within the graph database strongly differentiates this problem, even from the existing attempts at data structure optimization, as they focus entirely on graph structure [162, 163, 175]. The existence of properties is critical to our methodology, as it is not possible to ignore the schema of the graph when profiling simple operations as in existing works (e.g. profiling a GetVertex operation is not sufficient, since the schema of the vertex added will have a large impact on performance).

In-line with our view of routers as structures with AI capabilities, we learn approximations to $\bar{c}_{ij}$ and $\bar{f}_j$ independently at each router. That is, each router generates and executes sample queries according to the router's schema, and stores the resulting models for solving a problem akin to Problem (4.5), where we make the substitutions $c_{ij} \leftarrow \bar{c}_{ij}, f_j \leftarrow \bar{f}_j$, and $h \leftarrow \bar{h}$. Thus we are using average router performance as a model for these functions, something that is facilitated at finer grains as we increase the number of routers (recalling that it is the router abstraction and the implementation of routers with goroutines that allows us to fathom router numbers in the tens of thousands, rather than the the relatively few shards used in typical thread-focused graph computing).

4.3.4 Experimental Results

We use random forests [176] to learn each of the $\bar{c}_{ij}, \bar{f}_j, \bar{h}$ approximations owing to their ease of use combined with their ability to capture non-linear interactions. To demonstrate the efficacy of the

method, we construct an artificial graph system consisting of four vertex types; Person, Product, Location, and Mail (with 300, 400, 500, and 600 vertices of each type respectively). We split the graph by vertex type, such that a given router primarily contains a single vertex type. As we allow edges both within the routers and between them, each router also contains a number of duplicated vertex types copied from other routers due to external edges. We constructed the graph using the Erdos-Renyi model [177] such that average degree is given as in Table 4.2. We note here that we do not expect our method to require a graph substantially similar to the graph presented here-in, or even to partition by vertex type as we have done. Instead, this example was generated for ease of understanding and reproducibility. We have similarly designed the average between router degree to be much smaller than the within-router degree. This is because methods of graph partitioning typically involve minimizing edges or communication between partitions [155], and so we believe this a reasonable choice (note that this work does not deal with the question of graph partitioning, since it would need to be solved jointly with the assignment problem; such investigation is left to future work).

In terms of edge properties, we add two properties to each edge; a bivariate Gaussian with correlation ρ chosen $\sim \mathcal{U}[0,1]$ for each edge type. We thus demonstrate that queries interacting with multiple properties are not an inconvenience for the method.

Basic Operations

For the purposes of learning basic operations, we simply run many instances of the basic operation on each edge and vertex type described in the schema, for varying sizes of the graph (that is; the only input available is the total number of vertices and edges in a graph). In this instance, we are not training specifically on the graph described in Table 4.2, merely graphs of different sizes with the same vertex and edge types. For this reason, the learning of $\bar{h}$ functions, which we do independently for each representation type (RDF and MAL in our example), can be performed independently of any particular graph, only depending upon the schema. This indicates that if we expect multiple graphs with the same schema, we need only learn our approximations $\bar{h}$ once for each representation. In contrast, the methods described below require training on each individual graph.

Table 4.2: Average out-degree for each vertex combination

	Person	**Product**	**Location**	**Mail**
Person	32	0.08	0	0.06
Product	0.1	26	0.12	0.04
Location	0.01	0.02	255	0.01
Mail	5	3	8	100

Complex to Basic Count Mapping

In order to learn $\bar{c}_{ij}$, we generate a number (in our experiments, approximately 10,000) of single-step test queries. These queries are of various types (e.g. filter a list of edges, filter in-bound from a list of vertices, filter out-bound from a list of vertices etc). In order to ensure that our methodology can be applied more broadly, we generate these test queries by randomly sampling vertices and edges from each router (the number of which is chosen uniformly from zero to the number of edges/vertices in the router). We then generate test queries randomly designating a property on the sampled graph object, such that we are searching for edges that are greater than or less than the property at the sampled edge. Clearly this technique will generate queries that span the range of allowable property values, without any need to know in advance the distribution of any given property on the graph. We tag each query with a unique identifier, and track that identifier as it results in the execution of basic operations (that is, for any given complex operation, we know exactly where basic operations occur, which operations they are, and how many of them occur).

We then use a random forest to learn this mapping, such that the test input includes the number of vertices or edges we wish to start the query from. In addition, we include the number of vertices shared between the source router (i.e. the router where the query is first initiated) and the other routers. We found that the inclusion of the latter was key to obtaining random forest models with very high accuracy. We observed that the random forest models quickly learned to become very accurate. We note that this is not a case of overfitting; all experiments used half of the samples for training, and half for evaluation. Indeed, the nature of our sampling methodology means that the sample space of possible queries is extraordinarily large. With $R^2 \rightarrow 1$ for many mapping functions, we observed the power of local learning, where the models quickly pick up on router characteristics.

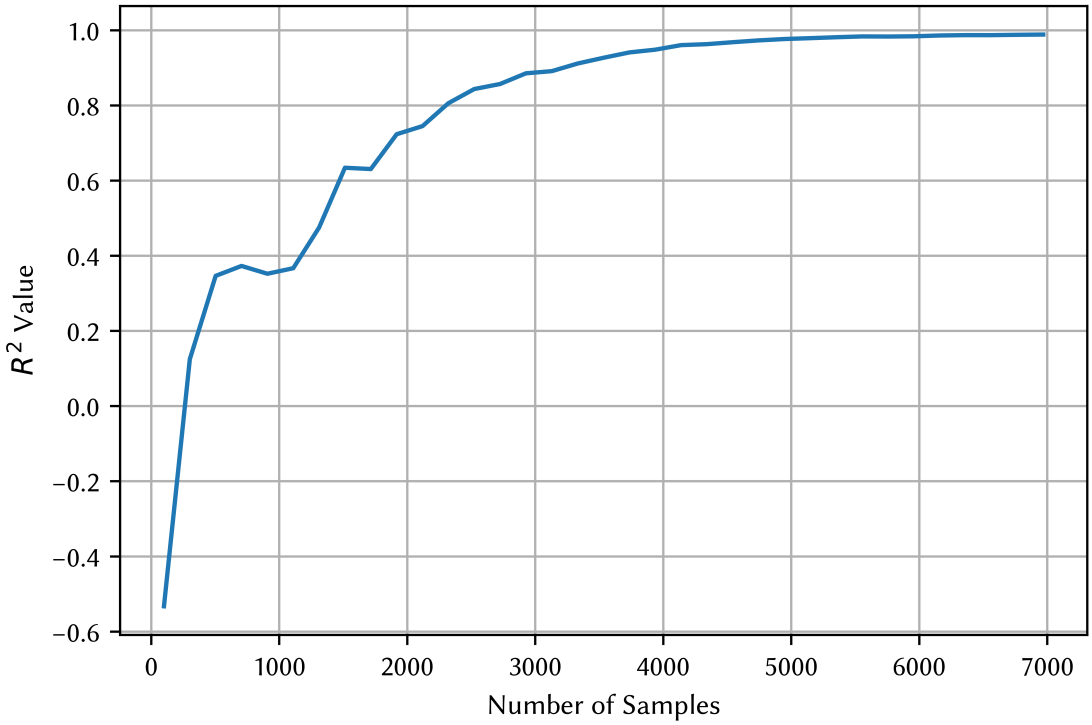


Figure 4.13: R^2 for learning vertex output and router locations given list of vertices and filtering conditions.

Learning Filtering

The method for learning filtering (that is, the number of objects that will survive a query) is very similar to that described above. However, here we additionally use the parameter values of the queries (as they relate to the vertex and edge properties examine in each query) as inputs to our random forest models. In addition to the relationships between the complex and basic operation tags, the system also keeps track of how many graph objects survive each filtering operation. As in the learning of $\bar{c}_{ij}$, we quickly obtain $R^2 \to 1$. Of course, we expect that the difficulty of learning the filtering mapping will increase as the number of variables that we perform filtering on increases. Nonetheless, as demonstrated by Figure 4.13, in this example the random forests are capable of learning the filtering mapping approximation to a very high degree of accuracy, given sufficient samples.

The Representation Problem

We use all the above methods to solve the representation problem. This problem involved sending 100 random two-step queries (that is, each query had an output that had to be fed into the next stage) to each of the four routers (as described by Table 4.2). We ran the queries in each of the sixteen possible sets of representation choices (in order to demonstrate the fully optimal solution), and then fed the query specifications to our learned models of $\bar{c}_{ij}, \bar{f}_j, \bar{h}$. We observed very encouraging results (as seen in Figure 4.14); using our machine-learning models gave us representations that on average were only 14% slower than optimal, with the average router representation being more than 13 times slower than the optimal. As optimal representations run the gamut from all-MAL and all-RDF to everything in between, the value of an approach that gets a near optimal solution is particularly important given that the consequences of incorrect assignment can be so drastic. Note that finding the true optimal solution requires running the set of jobs with every possible combination of router assignments. As mentioned previously, this requires an exponential number of router representation tests. As the size of the jobs increases, it becomes infeasible to look for an optimal representation through brute force (and to do so is redundant, as completing the job once is the purpose of the execution on the database). In contrast, the SmartGraph method will scale with the machine learning models used to learn $\bar{c}_{ij}, \bar{f}_j, \bar{h}$, and in many instances will not need to be retrained at all (e.g. if we keep the same graph and move from two-step to three-step traversal queries). Note again that despite our use of a surrogate based formulation (where we did not consider concurrency in our formulation, despite the SmartGraph system being specifically designed to take advantage of concurrency), we get very promising results that were able to drastically reduce *concurrent* runtimes.

4.4 Discussion

In this chapter we demonstrated the details of the SmartGraph, a new system for constructing graph databases, and detailed many of its important features. We showed how the Buffered Channel Management component of the SmartGraph facilitates high levels of automatically managed concurrent execution of requests, while retaining the ability to manually control asynchronicity to any level a user desires. We demonstrated how multiple routers and graph representations can be

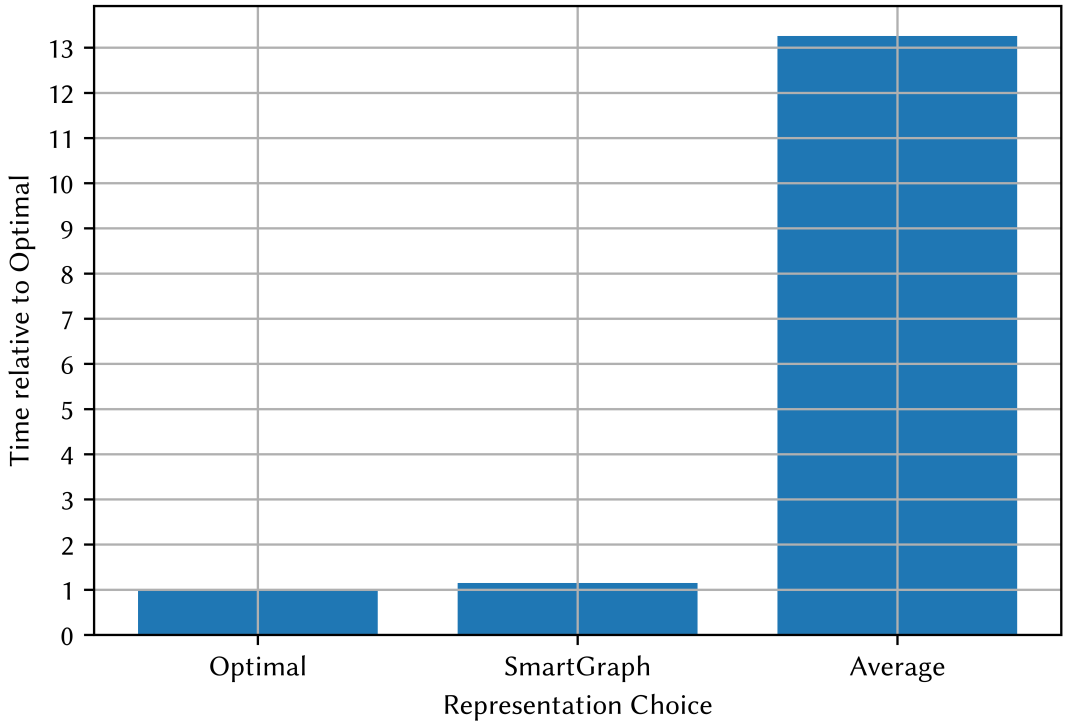


Figure 4.14: Representation Problem on our Test Cases.

of great benefit, and explored this further in the representation problem. We also used the representation problem to demonstrate how giving artificial intelligence to "router-like" subgraphs can be used to solve highly complex problems. We showed that the representation problem is an exponentially large combinatorial optimization problem, but that it can be solved to near-optimality by having each router learn machine-learning model representations of themselves and their neighboring routers. By adding artificial intelligence at the "edge" of graph database technology, we increased the computing capabilities of the graph database, and continue to muddle the space between graph computing and graph databases.

Chapter 5

The Computational Graph Query

Graph querying is often thought of as a static process for simply reading and interpreting graph structured data. We argue that recent innovations in dataflow and computational graph systems allow us to extend graph querying to actually *executing* a range of algorithms, including optimization problems that rely on graph structured data as inputs to their cost functions. We demonstrate how this approach can be incorporated into the SmartGraph system detailed in Chapter 4, and use the problem of recommendation systems as a practical example of its use. In this chapter, we bring together many of the concepts, methodologies and systems developed elsewhere in this dissertation to achieve our goal of machine learning with graph structured data.

5.1 Introduction to Queries and Query By Example

In graph databases, as in relational databases [178] and other forms of NoSQL databases [179], the purpose of querying is to summarize data within the database, or to extract information that meets a particular need (e.g. "find all users who executed transactions greater than ten-thousand dollars in the last 4 days"). In the relational database setting, these results are typically presented in a tabular form, matching the structure of the relational database. Graph queries typically result in graph structures, such as subgraphs, lists of satisfying vertices, or lists of satisfying edges. In some cases, results may also be presented in tabular or scalar forms (such as when queries relate to retrieving particular properties of graph objects without reference to graph structure).

As discussed in Chapter 4, methodologies for graph querying typically fall into two categories; path queries [170], and pattern matching [174]. In pattern matching [104, 180], a subgraph is

supplied, with exact or approximate [181–184] matches within the overall graph returned as results. A path query on a graph typically has a form $I \overset{cond.}{\rightarrow} J$ where we are seeking paths from graph object set I to (potentially unknown) graph object set J that must satisfy conditions *cond.* along the path. This is the approach taken by many prominent graph querying algebras such as Gremlin [146]. It is also the approach taken by the SmartGraph in defining its filtering and operations (see Chapter 4 for more details) due to it being more mathematically compatible with the representation problem, concepts of Bayesian inference and deep learning (such as ancestral sampling and computational graph execution), and the Computational Graph Query (CGQ) as outlined in this chapter.

In the graph database context, the goal of "querying by example" (also sometimes referred to as "reverse engineering", or "the reachability problem") [185–187] is to learn queries that return certain vertices (or more general graph objects). For the most part in the existing literature, query by example is explored from a theoretical perspective as part of a larger formal analysis of graph querying methodologies (see [171] for a recent overview of the field) in the context of graphs with relatively fixed structure and properties. The graph is viewed as strictly representing data, as opposed to representing the expression of a problem, or algorithm, etc. However, it is clear from recent work in deep learning, where algorithms *are* commonly processed as (computational) graphs [55, 188, 189] (with idea of *expressing* algorithms as graphs being well-established [190]), that graphs are very effective structures for representing computation in addition to network structured data. In this work, we are loose in our use of "by example", as referring to either specific vertices or edges (as in the recommendation problem explored in Section 5.7.3), or, in a more general sense, as "query results that give high utility" (where the meaning of "utility" in this context is made clear in Section 5.2 below).

The primary methodological contribution of this chapter is to demonstrate a method for performing interpretable queries that can be used to execute algorithms and solve optimization problems with any graph structured data as input (in contrast to the more traditional embedding techniques discussed in Chapter 2 and Chapter 3). This is accomplished by taking some key ideas from Chapter 3 significantly further. In particular, we allow the SmartGraph to store variable parameters (similar to the Variable tensor objects in TensorFlow [55]) that can be used to define complex expressions, but also to interpret the graph by executing path queries through it during algorithm or optimization execution. That is, we read the graph as an input into our algorithms

not by summarizing the graph into factors or apriori embedding it (or its constituent vertices etc.) into a vector space, but by exploring the graph during execution of analtyics or optimization via path querying.

5.1.1 Filtering Implementation in the SmartGraph

In Chapter 4, we described the implementation of querying and filtering operations in the SmartGraph only as they related to the representation problem. Here we give more details about the implementation of the filtering and querying functionality of the SmartGraph system, as required by CGQ.

In the SmartGraph, we define functions classified by a direction (when a direction can be defined) and triples of InputType − CentralOperation − OutputType, such as representing a filtering function VertexInput − EdgeFilter − VertexOutput that takes a set of vertex IDs, finds outbound (or inbound, if specified) edges and performs the filtration step upon these edges (i.e. applying the filtering conditions), and then returns the set of vertices on the other side of the surviving filtered edges. We can easily chain complex filtering operations together as a sequence of such requests, and generate a valid path query as a result. For a given filtering operation, each part is executed before the next part begins; for example, all first stage vertices in a VertexInput − EdgeFilter − VertexOutput query step are examined before the second stage of querying the edges is asked to begin (though as discussed in Chapter 4, these filtrations may execute asynchronously if the first stage vertices are duplicated in other routers). We can also add additional complexity in the middle step, e.g. if the CentralOperation is actually a more complex operation (e.g. a different complete path query that takes the correct input type and generates the correct output type could be put in the place of a simple vertex or edge filter, allowing us to have arbitrarily asynchronous control over the execution of queries).

The complex operation implemented in the SmartGraph that is of key importance to this chapter is the InputType − Split − OutputType series of filtering functions. This series of functions allows a single filtering input request (and its input set) to probabilistically generate different types of CentralOperation requests (e.g. satisfying different filtering requirements) given a probability simplex along with the associated CentralOperation objects. Consider two E CentralOperation objects, E_1 and E_2 that represent "friendship" and "family" edge label filtering constraints respectively

(e.g. in a social network). Then for $\alpha = [0.3, 0.7]$, 30% of the edges (chosen uniformly at random) outbound from the input vertex set will be filtered by E_1, and the remaining 70% by E_2, resulting in two different sets of vertices (friend vertices corresponding to E_1, and family vertices corresponding to E_2). Thus by changing the probability simplex, we can change the nature of the query step that is executed. We can do this for some (or all) query steps in a full path query. This is a central tenet of the CGQ methodology.

5.2 Queries as Functions on a Graph

Consider a property graph $G = (V, E)$ with properties on both vertices and edges. We allow for the properties to be *variables* ϕ or fixed data. Let O denote set of permitted query operations, and let F_O denote the set of all possible queries expressed by the operation set O. Then the "query by example" problem outlined in Section 5.1 can be formulated as the optimization problem

$$\max_{Q \in F_O} \; f(Q, S) \tag{5.1}$$

where $f : F_O \times S \to \mathbb{R}$ denotes the utility of a query Q starting from source nodes S. The utility function in a typical query by example problem is determined by some measure of accuracy on the match with the "examples". Here we allow the problem to be more general, where the utility function value might be determined by the "quality" of the returned graph object(s), with no actual "examples" necessarily required. This optimization problem is computationally hard, and is typically solved using heuristics.

We show that a useful mechanism for expressing graph queries is to appropriately parameterize them and consider them as "functions" on a graph. While the ideas that we propose apply to a more general set of parameterizations, we focus on queries that are generated by parameterized probability simplexes (though we keep our notation as general as possible) that control choices made during the execution of path queries. In this manner, we see the direct relevance of the work completed in earlier chapters of this dissertation.

Suppose a query is defined by a parameter θ (where $\phi \subseteq \theta$ in the most general case) and let Θ denote the set of all possible parameter values. Let $Q(\theta)$ denote the complete return query array: the (not necessarily numeric) array of nodes and other graphical structures returned by the query

Q over the entire graph, corresponding to parameter $\theta \in \Theta$. We use the term array rather than set because loops involving repeated elements are allowed in the result. Let a single traversal path obeying the query rules of $Q(\theta)$ be denoted $Q_i(\theta)$. $Q_i(\theta)$ obeys all step orderings and filtering rules associated with the full query $Q(\theta)$, but only picks up a single object during each step: e.g. if $Q(\theta)$ is to get the neighbors of the neighbors of a vertex β, some $Q_i(\theta)$ will return an array containing a single neighbor of β, i.e. ζ, a single neighbor of ζ, and the edges connecting them.

In our solution methodology, which we call the Computational Graph Query (CGQ), we approximate $Q(\theta)$ by N traversals as follows:

$$Q_N(\theta) = [Q_i(\theta)]_{i=1,\ldots,N} \approx Q(\theta), \tag{5.2}$$

corresponding to parameter θ and the resulting path query. Observe in (5.2) that we define $Q_N(\theta)$ as an array of $Q_i(\theta)$. As each $Q_i(\theta)$ is itself an array, $Q_N(\theta)$ is therefore an array of arrays (that can be flattened to a simple array). This is explicitly formulated this way to emphasize that traversals may collect the same graph structures (e.g. pick up the same vertex) during multiple traversals, and that the collection of these repeated structures is to be emphasized in our formulation. E.g. if a vertex is picked up in multiple different traversals, we argue that it demonstrates the importance of the vertex; importance that would be diminished if we used sets and considered the vertex only once.

Next, we approximate the optimization problem (5.1) by a sample-based approximation:

$$\max_{Q \in F_O} f(Q, S) = \max_{\theta \in \Theta} f(Q(\theta), S) \approx \max_{\theta \in \Theta} f(Q_N(\theta), S). \tag{5.3}$$

Note that the traversal samples in $Q_N(\theta)$ are in part generated by discrete probability distributions of the form $\mathbb{P}(D = i) = p_i$, since we are using probability simplexes $p \in \theta$. We want to choose parameters of the simplexes p to maximize the utility. The samples of D can be generated by setting $D = \arg\max\{-\log(p_i) - \log(-\log U_i)\}$ where $U_i \sim \text{unif}[0, 1]$. However, the $\arg\max$ function is not differentiable in p; hence one cannot use gradient descent to compute the optimal p.

But this is exactly the problem dealt with in Chapter 1 (and in Chapter 3). We replace the discrete distributions of the query operations by their continuous approximation counterparts (e.g.

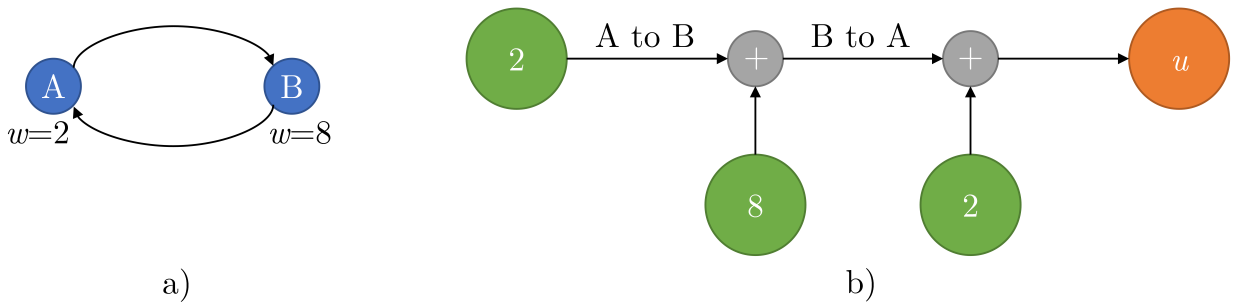


Figure 5.1: Unrolling of a query on a two node graph database graph (a) into a small corresponding computational graph (b).

through the Concrete distribution, using unnormalized α rather than the normalized p equivalents), allowing us to perform gradient-based optimization, such as Stochastic Gradient Descent, in computing an approximate solution to (5.1).

5.3 A Curse of Dimensionality in Computational Graph Queries

Consider a graph database with two connected nodes, A and B as shown in Figure 5.1a, where blue nodes are actual graph database structure. These two nodes have directed edges to one another, and a vertex property w. Consider the trivial query "starting from vertex A, sum all values of w for all two hop paths, and take the average over all such paths". Because of the cycle, A is a vertex within two hops of itself, and so the single possible path query taken goes $A \to B \to A$. Let u be the result of executing this simple query on Figure 5.1a. Then the corresponding computational graph, resulting from unrolling the query on the graph database into a computational graph, is given by Figure 5.1b. Green nodes are input to a computational graph, black edges denote the flow of data (where we have indicated when edges also correspond to traversals in the graph database), grey nodes are operations, and orange nodes are outputs of a computational graph. By unrolling, we have formed a directed acyclic computational graph from a graph database that clearly contains a cycle.

Similarly, we can unroll more complicated path queries into directed acyclic computational graphs. Consider a graph database differing from Figure 5.1a only by an additional fully connected node, C, shown in Figure 5.2a. In 5.2b, we demonstrate how the addition of this single node results in a significantly larger computational graph (when we unrolled with the same query as

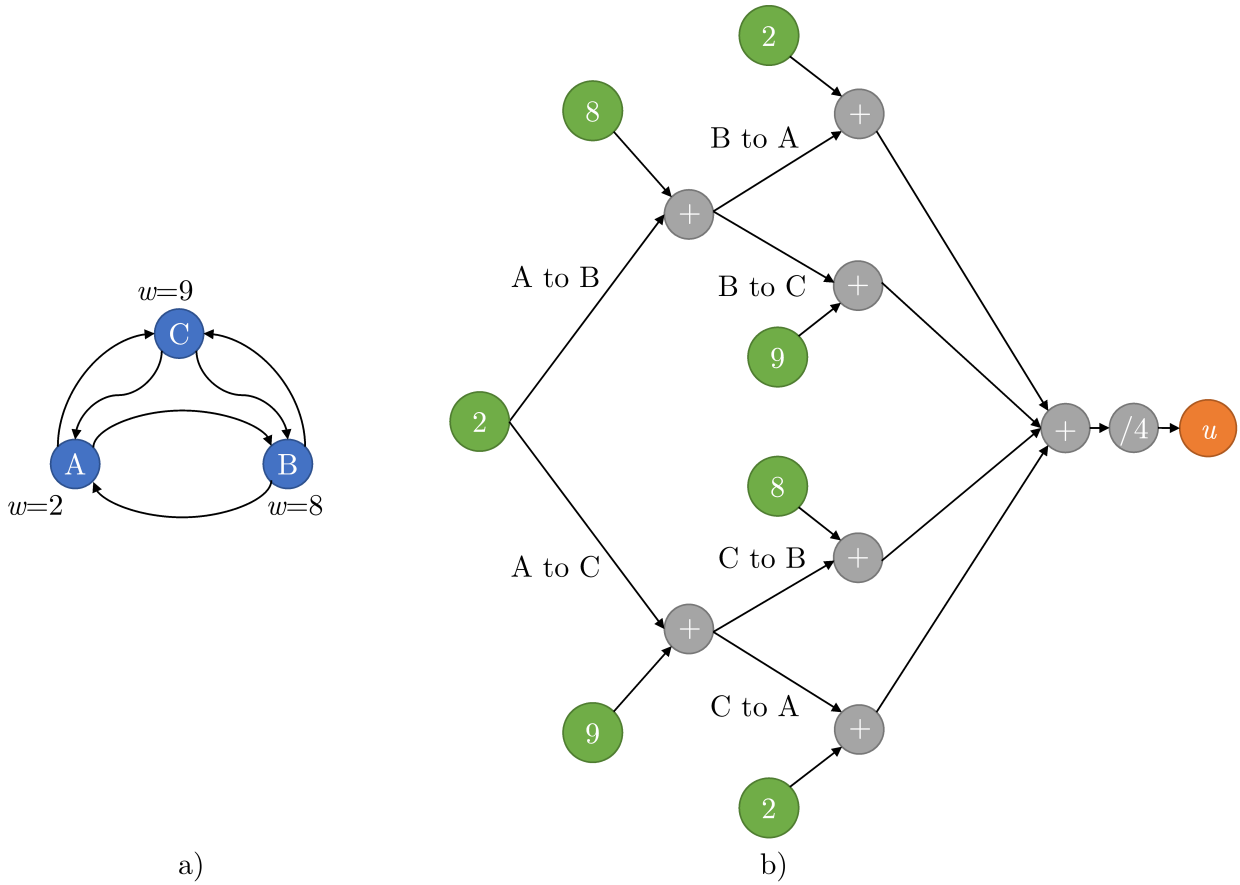


Figure 5.2: Unrolling of a query on a three node graph database (a) into a large computational graph (b).

used previously). By adding a single node, we have gone from one possible path satisfying the query to four, with a significantly larger computational graph as a result. This is the result of a very modest change. Consider if we instead doubled the number of hops in the query, or doubled the number of nodes in the graph database; we quickly encounter a case of exponential growth in the unrolled computational graph.

As discussed in Section 5.2, we are often interested in queries that involve discrete values or choices. As outlined in Chapter 1, we can use the Concrete distribution for this situation, where the use of the Concrete distribution replaces the request for taking an average over all paths. We demonstrate how the Concrete distribution would be used for a similar query (on the same three node graph database) in Figure 5.3. We label Concrete operations as nodes s_1, s_2, s_3 in the computational graph in Figure 5.3b. We assume these Concrete distributions are respectively

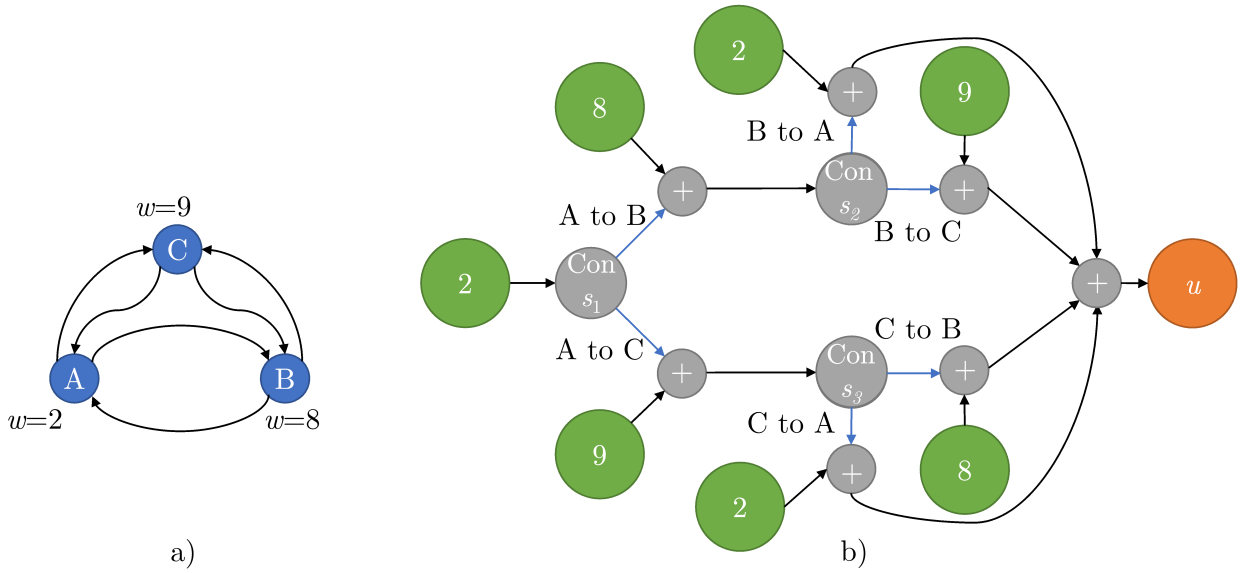


Figure 5.3: Unrolling of a query with Concrete traversals into a corresponding computational graph.

driven by parameters $[\alpha_1, \alpha_2, \alpha_3] = \theta$, but omit these parameters from the computational graph for readability.

The output of these nodes are weights on a simplex, represented as blue edges. The blue edges (that sum to one) are then *applied as a multiplication on all further calculations on that path.* This would most properly be represented in a computational graph as an operation of a multiplication by the relevant weight at the end of the associated path, however, such a graphical representation soon becomes too cumbersome and hard to interpret when considering sequences of decisions (the earlier a "decision" is made, the later it would appear in the computational graph).

Let the Concrete nodes generate Concrete samples $s_1 = [0.9, 0.1], s_2 = [0.7, 0.3], s_3 = [0.2, 0.8]$ (for purposes of illustration, we are overloading the notation s here to indicate both the node and the sample values generated by the Concrete distributions: running the same computational graph again would lead to different values). We assume that the simplex is ordered such that indices corresponding to edges that are drawn higher are lower (i.e. s_1 in Figure 5.3b is ordered as $[A \to B, A \to C]$). The value of u in this setting would then be calculated as follows:

$$u = 2 + 0.9[8 + 0.7(2) + 0.3(9)] + 0.1[9 + 0.2(8) + 0.8(2)] = 14.11, \tag{5.4}$$

where the blue terms indicate our Concrete elements that multiply the rest of the corresponding

path. Since we have a sequence of decisions, we have multiplications of the blue terms by other blue terms. This nesting of multiplication is why the true computational graph is not as interpretable as that given in Figure 5.3b. We have written the expression for u in the same order as our method of indexing s, with later decisions nested in earlier decisions. In addition to the nesting, note that this formulation means that all values going forward are multiplied by the Concrete sample element, but inputs to the Concrete nodes pass through with their full value.

The Concrete distribution works fairly well in practice for many applications. However, it has a fatal flaw in any setting with considerable fan-out (such as CGQ). We observe that in (5.4), there is a calculation associated with every path (from the first "2" input node to u) through the computational graph, each of which correspond to a path in Figure 5.3a that satisfies the query.

We emphasize the issue by showing how queries on graph databases "unroll" into computational graphs in the CGQ setting (where executing the computational graph runs the query on the graph database). We use the term unrolling because a simple query on a simple graph database results in a long and wide feed-forward computational graph that only grows as the complexity of the query or graph database increases.

This means that the Concrete distribution has a problem when used with CGQ; an exponentially growing computational graph associated with a query means an exponentially growing computation, since there are computations associated with every edge. For gradient descent, this occurs in both the forward-pass (where we want to calculate the value of the utility node u, and values for each intermediate node), as well as in the backward-pass (where we want to back-propagate gradients into the parameters $\alpha_1, \alpha_2, \alpha_3$ that control the samples generated by the s_1, s_2, s_3 Concrete operations).

5.4 Computationally Tractable Approximations to the Concrete Distribution

We are thus motivated to develop an approximation to the Concrete distribution that does not suffer from this issue of exponential growth within the CGQ setting. We are not the first to develop an approximation to the Concrete distribution, however, we are the first to do so specifically for reasons of computational complexity.

The Concrete distribution allows us to replace discrete variables with continuous approximations over a simplex, facilitating the use of gradient descent methods in computational graphs. However, many models will not allow us to make evaluations in a continuous space. Consider for instance, either going left or right in a maze. We cannot go both left and right simultaneously (nor can we go $x = [0.99, 0.01]$, 0.99 left and 0.01 right); we must instead commit to a direction and follow it before we can evaluate the utility of our decision. This trivial example demonstrates a problem with the Concrete distribution; it may be impossible to evaluate the objective or utility function (or even know what the appropiate function may be) in some settings given an input over the simplex. Of course, this does not cause an issue in all settings; it may be possible to directly evaluate a simplex sample in some settings by taking an expectation over the simplex, e.g. as we did in (5.4).

In situations where direct evaluation is not possible, the Straight-Through Gumbel-Softmax Estimator [13] (STGSE) is used. Let x be a Concrete sample on the simplex Δ^{k-1}, parameterized by some set of variables θ. Then a "hard-sample" is given by $y = \text{one-hot}(\arg\max(x))$, where we define $o^* \in \{1, \ldots, k\}$ as the index of the largest element of the vector x. This one-hot setting means that truly discrete choices are made (because no weight is given to choices other than the argmax), always allowing for evaluation of the utility function.

A side effect of the STGSE is that it removes the exponential growth in the *forward-pass* of a CGQ iteration. Consider Figure 5.3 once more, where we assume that the Concrete nodes output STGSE samples. We therefore have the following:

$$\begin{aligned} s_1 &= \text{one-hot}(\arg\max([0.9, 0.1])) = [1.0, 0.0] \\ s_2 &= \text{one-hot}(\arg\max([0.7, 0.3])) = [1.0, 0.0] \\ s_3 &= \text{one-hot}(\arg\max([0.2, 0.8])) = [0.0, 1.0]. \end{aligned} \tag{5.5}$$

This means that we choose the path $A \to B \to A$, and our calculated value of u is as follows:

$$\begin{aligned} u =& 2 + 1[8 + 1(2) + 0(9)] + 0[9 + 0(8) + 1(2)] = 12 \\ =& 2 + [8 + (2)] \\ =& 12. \end{aligned} \tag{5.6}$$

Observe that many of the blue values are zero. This means the corresponding paths are multiplied by zero, and therefore, all nodes along that path will have value zero (excepting when combined into the utility at the end of the calculation). This means that we do not need to *calculate* the values of the nodes along these paths, because they will all be zero. As a result, the STGSE has eliminated the exponential complexity of the *forward-pass*, since we only calculate values along the path taken in the computational graph according to the sequence of decisions made at the Concrete distribution nodes.

In the STGSE, a single forward pass is computed quickly at any Concrete distribution node in the simplex Δ^{k-1} through the hard-sample approximation above. Because we still cannot differentiate with respect to a max, STGSE uses the gradient of the standard Concrete distribution when back-propagating:

$$\nabla_\theta y \approx [\nabla_\theta x_1, \ldots, \nabla_\theta x_k]. \tag{5.7}$$

Recall that in the backward-pass of gradient descent in a computational graph that the chain rule is used starting from the end node (e.g. the utility) with a gradient of $\nabla \text{end} = 1$, with gradients back-propagated as follows:

$$\nabla r = \sum_{v \in out(r)} \nabla v \frac{\partial v}{\partial r}, \tag{5.8}$$

where $out(r)$ is the set of all output vertices from the computational graph node r. Unfortunately, despite all nodes in the non-chosen paths of the STGSE approximation having value zero, they do not necessarily have a zero gradient. This means that back-propagation must propagate throughout the entire graph, and not just the chosen forward path. Though the STGSE reduces computational complexity in the forward pass, it does not do so in the backward pass, and the order of complexity of the overall calculation is thus the same as for the standard Concrete distribution. The primary purpose of the STGSE, then, is in allowing the evaluation of a cost function that might otherwise be undefined (and not necessarily in computational simplification).

We introduce the Straight-Back Gumbel Softmax Estimator (SBGSE). Like the STGSE, a single forward sample pass of the SBGSE is calculated by using a $y = \text{one-hot}(\arg\max(x))$ approximation. This means that the forward pass of SBGSE is calculated exactly as for the STGSE, and retains the reduced computational complexity of the forward pass.

The challenge is, therefore, in reducing the complexity of the backward-pass. In addition to the STGSE-like one-hot approximation during the forward pass, the SBGSE also uses a modified backward-pass. The idea here is quite simple; we only back-propagate along the chosen path, and do not compute gradients along any path that has been zero multiplied (i.e. by a zero element of the one-hot vector). SBGSE therefore uses the following gradient approximation for a Concrete node on the simplex Δ^{k-1}:

$$\nabla_\theta y \approx [0,...,\nabla_\theta x_{o^*},...,0] \neq \nabla_\theta x, \tag{5.9}$$

where o^* is the maximizing index of the one-hot forward sample. Note the differences between (5.9) and (5.7). Rather than approximating all gradient elements of $\nabla_\theta y$ by the corresponding gradient in $\nabla_\theta x$, we approximate using only the gradient of the element that corresponds to the hard-sample, with the rest set to zero. This is equivalent to never calculating or storing the zero gradients at all (and in fact, we do not do so in the SmartGraph), because a zero gradient makes no contribution to the calculation of earlier gradients during back-propagation (see that there is no contribution to earlier node r in (5.8) when $\nabla v = 0$).

To briefly demonstrate the backward-pass (and forward-pass) differences between the Concrete distribution, the STGSE, and the SBGSE, we zoom in on decision s_2 from Figure fig:cgq-unroll2b, assuming that $s_1 = [1.0, 0.0]$. This is shown in Figure 5.4. We do this to give more insight into the Concrete nodes, as well as the complexity of showing back-propagation through the full computational graph of Figure 5.3b.

In Figure 5.4, we show the forward- and backward-passes for the Concrete distribution, the STGSE, and the SBGSE. We observe that the STGSE shares a backward-pass with the Concrete distribution, and a forward pass with the SBGSE. We outline in red the path taken (from B to node C and its $w = 9$), and include the $2 + 8$ as the utility coming from the previous $A \to B$ traversal. The z correspond to the Gumbel samples as outlined in (1.2), with $\alpha = [0.3, 0.7]$ a randomly selected initialization, and c_A/c_C corresponding to the node weights of A and C respectively. Since many of the operations (e.g. log, plus, division, softmax) are vectorized, we separately include the forward- and backward-passes of each element above and below the operation nodes (e.g. $\log(0.3) + (-0.49) = -1.69$).

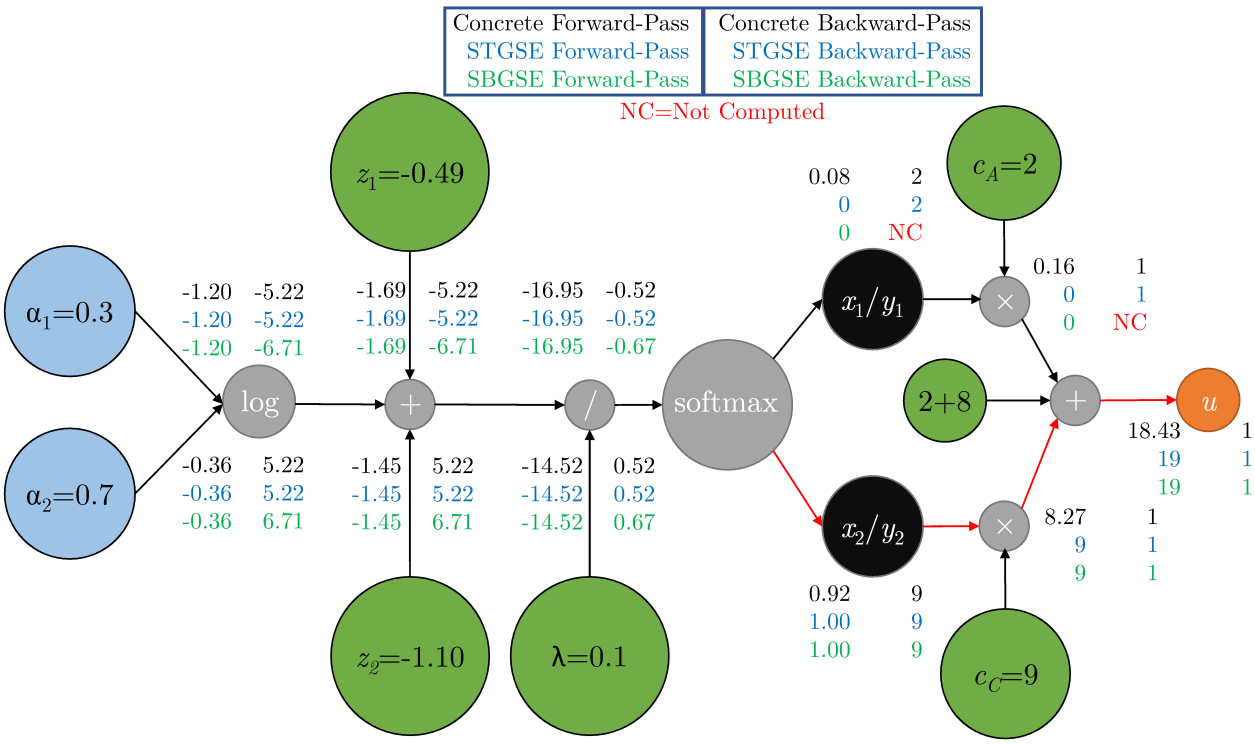


Figure 5.4: Comparison between Concrete, STGSE, and SBGSE in forward and backward passes.

We note that in the SBGSE, we do not compute the backward-pass for the $B \to A$ path, as it is zero (which, as argued above, means it is not necessary to calculate). In contrast, STGSE does calculate (non-zero) gradients along this path, and these ultimately affect the gradients determined at the α nodes. SBGSE in general does not need to back-propagate along any zeroed out path.

In order to analyze differences in computational efficiency between the SBGSE and the STGSE in a more general setting, we consider an extended computational graph in Figure 5.5. Here we consider a sequence of l decisions $d_i, i = 1 \ldots l$, each from k options, where the available k options differ depending upon previously made decisions (and correspondingly, the utility of the decisions is dependent upon the entire sequence, which we assume is non-zero for all possible sequences of decisions). In order to aid visualization, we are ignoring the computational complexity of the "Concrete" operation nodes that might otherwise appear in the computational graph. We introduce the node θ, connected to every other node, such that each node makes a fan-out decision that is dependent upon θ i.e. some function of θ describes the α terms used during sampling at each node.

Consider a hard sample of decisions made in this setting. As indicated by the dotted line in Figure 5.5, the forward pass of both STGSE and SBGSE moves along only $l + 2$ edges. This single

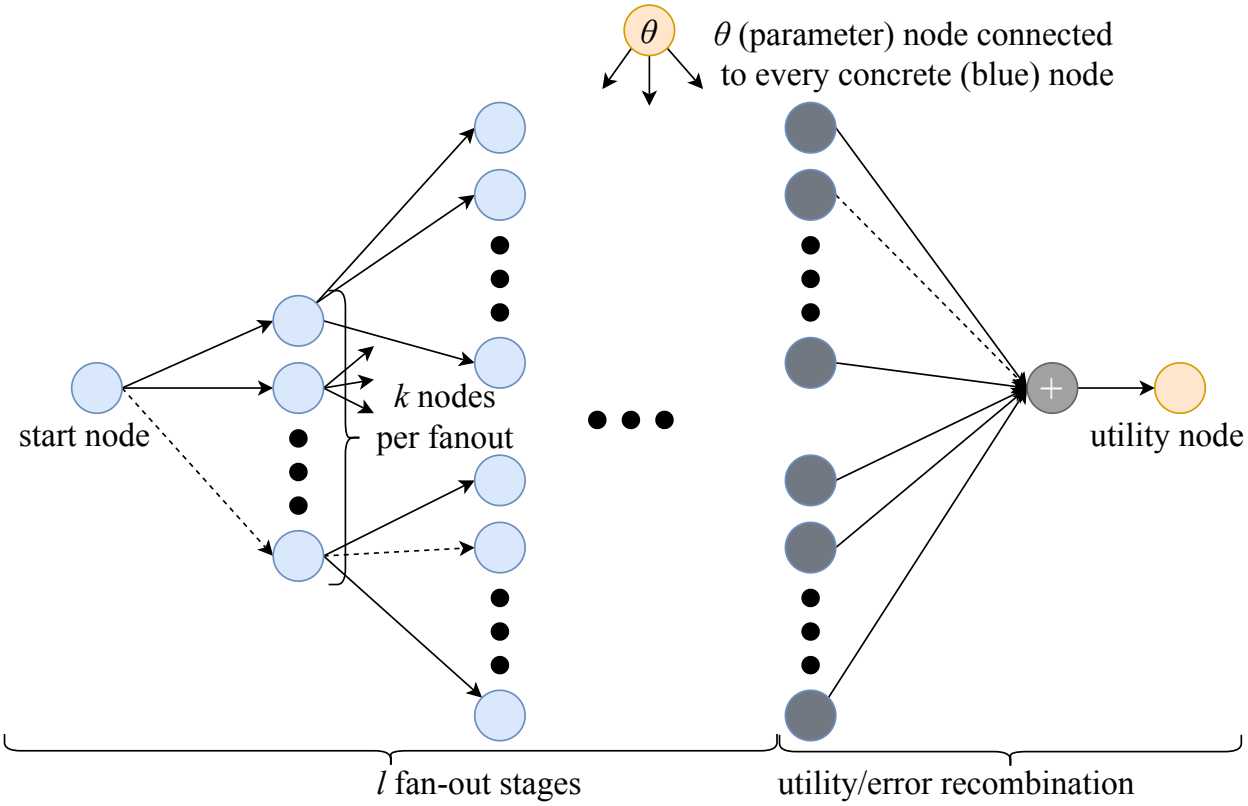


Figure 5.5: Large-scale computational graph of a series of history dependent discrete decisions.

forward sample can thus be computed very quickly for both STGSE and SBGSE. Consider back-propagation on this computational graph in the STGSE setting. At each node of the computational graph in the fan-out stages, a decision between k alternatives is being made. As we demonstrated in Figure 5.4, a zero value at a node in a hard forward-pass does not mean the gradient is non-zero. With non-zero gradients potentially everywhere, naive back-propagation in SBGSE would need to compute an enormous amount of computations. In contrast, we show in Theorem 5.1 that the complexity of SBGSE is considerably more forgiving:

Theorem 5.1. *Back-propagation of the Straight-Through Gumbel Softmax Estimator in the l stage k alternative fan-out is $O(k^{l+1})$, whereas the Straight-Back Gumbel Softmax Estimator is $O(l)$.*

Proof. There is a computation associated with every edge during back-propagation of the STGSE, and so the complexity of back-propagation is $O(E) = O\left(\sum_{i=1}^{l} k^i + \left(\sum_{i=0}^{l+1} k^i\right) + k^{l+1} + 1\right) = O\left(k^{l+1}\right)$, where the terms correspond to edges from Concrete nodes to other Concrete nodes (to the k^{l+1} grey recombination nodes, after the lth fanout), edges from θ to all Concrete nodes, edges from the recombination nodes, and the edge to the final utility node.

In the SBGSE, there is instead a backward calculation associated with every edge *traversed by a walk* through the computational graph, and a calculation for the connections of these nodes back to θ. A walk of SBGSE traverses l Concrete related edges, 2 edges in the recombination stage, and l edges from the Concrete nodes back to θ, resulting in $O(l)$ complexity. □

Note that $O(l) \ll O(k^{l+1})$, for even moderately sized graphs. Given the large size and high degree of structural complexity present in many real-world graphs (often exhibiting the "supernode" phenomena [191] where some vertices have extremely high numbers of outgoing edges, resulting in high values of k), and because many graph queries have a number of important stages (resulting in high values of l), it is clear that STGSE will break down due to the exponential complexity. Although naive back-propagation is typically very efficient in most applications (because the widths of the network layers are controlled), it is hampered by the fan-out of the scenario described herein. When we run graph queries, we are soon looking at neighbors, then neighbors of neighbors, their properties, ad infinitum. Thus running a query on a graph in a graph database unrolls into a computational graph similar to Figure 5.5. This was observed in Figure 5.3b, even for a very simple query on a simple graph. We argue Figure 5.5 and its complexity results are, therefore, a strong basis for analyzing the computational efficiency of CGQ.

5.5 A Self-Directing Multi-Sample Approximation

Theorem 5.1 is not a complete picture for the complexity comparison. The back propagation of STGSE provides gradient information to *all* parameter components of θ, whereas SBGSE only propagates gradient information to those variables involved in the nodes through which the hard-sample traversed. As discussed in Section 5.2, we are thus motivated to use a multi-sample approximation. The obvious benefits of this are getting more accurate gradients with respect to the parameter components of θ, as well as getting gradients related to more of the components.

Let there now be N SBGSE traversals. In this situation, we approximate each Concrete sample gradient as outlined in (5.9). We are approximating the gradients of individual hard-samples with the gradients of the individual Concrete samples (or at least, the gradient of each samples' most significant element). It is common in gradient approximation methods to use multi-sample approximations to get a more accurate estimation of the gradient of a parameter (either in terms

of reduced bias, variance, or both). It would appear that we are still using a strictly single-sample approximation in the SBGSE; and this is true, when we are considering the individual traversals in a vacuum. However, recall that the actual parameters of interest are not the paths themselves (which are related to the series of y along each path), but the parameter θ that controls *which* paths are taken through the graph. If the gradient of the utility function with respect to parameter θ contributed by walk n of N total walks is given by $\nabla_{\theta^n} u$, then we simply define the following:

$$\nabla_\theta u \approx \frac{1}{N} \nabla_{\theta^n} u. \tag{5.10}$$

This means that, by virtue of the back-propagation process, any paths that share nodes in the computational graph will contribute to the gradients of those nodes. For example, multiple paths in Figure 5.3 that traverse $A \rightarrow B \rightarrow C$ will all contribute to the gradients of the α variables corresponding to the Concrete nodes along this path. The same holds for any variables that are shared between traversals (this will involve traversing the same parameter node in the computational graph, but the corresponding paths in the graph database may be entirely disjoint). In this sense, the STGSE is actually a multi-sample approximation.

This is not the first such notion of a multi-sample approximation of the Concrete distribution; indeed, one of the original Concrete distribution papers [12] discussed the possibility of a multi-sample approaches (using NVIL [192] and VIMCO [193]), though their results focused on a single-sample approximation of the gradient. The other [13] implemented such multi-sample approximations, but the reasoning behind the multi-sample approach in those papers was very different to our use-case, in that the primary focus in those papers was on removing bias and lowering variance of the calculated gradient (a common use for multi-sample approaches in many computational graphs).

The goal of SBGSE is not to reduce bias or variance, but to develop a continuous approximation to categorical random variables that has a computationally tractable backward pass in the case of extreme fan-out. A key advantage of our multi-sample approach is that it is inherently selective based upon the "importance" of the associated parameter node in the computational graph. To see this, consider that our optimization of (5.3) drives our paths towards increasing utility. This means that a variable that is only traversed by a single path out of N is likely relatively unimportant. In this case, little effort is spent on ensuring the estimated gradient of this variable is

highly accurate. However, if this single traversal indicates there is perhaps benefit to changing the corresponding variable, the next generation of traversals will likely include more paths through this variable, allowing for a more accurate gradient estimation. We have therefore devised a multi-sample gradient approximation method that automatically devotes higher levels of computational effort to accurately estimate gradients of variables of higher import.

Through the use of multiple samples, we increase the chance that any given traversal will interact with a particular variable, and thereby back propagate information from the utility function. We generate a number of "random walks" according to some input graph query (that are actually dependent upon and differentiable functions of parameters of interest θ). The use of random walks and traversals to approximate graphs is a well-known technique in graph computing [112, 194, 195], but these approximations are typically used for embedding or calculating factor terms, as opposed to the SBGSE, which is changing its exploration and meaning with each iteration. SBGSE, then, can be viewed as a method combining back-propagation with random walks on graphs, where we *only back-propagate along the paths we have taken during traversals.* Though we do not theoretically investigate the question of "how many traversals" do we need for SBGSE to perform effectively, our empirical results in Section 5.7 and Section 5.7.4 suggest that the answer is "not many".

5.6 CGQ Optimization

Thus far we have focused on single iterations of a CGQ (where we use the term iteration standard to optimization methods). To gain an understanding of optimization in the CGQ context, consider a computational graph resulting from the execution of a single query on a graph in a graph database. This computational graph consists of data (i.e. from the graph database objects), as well as variables (in our examples thus far, coming from the Concrete distribution sampling procedure and its α values, though such variables can also be stored within graph database objects in the SmartGraph system).

As a result, we could take this computational graph as static, and optimize the variables until convergence, as in systems like TensorFlow [55]. Suppose that the computational graph corresponds to only a single querying procedure (even if that single query procedure involved many traversals). This means that even if the parameters θ change, the traversals are constant (for that iteration).

Therefore running many iterations of gradient descent on this first computational graph could lead to wildly inaccurate results, as we would have α parameters that no longer correspond to the paths as given in the unrolled computational graph.

We have previously discussed our goal in enabling deep learning capabilities via querying, and that this is accomplished by reading the graph during optimization. To be precise about our meaning, CGQ does not view the graph database as a static object, and it does not use a static computational graph. Instead, the computational graph used by CGQ is *dynamically generated by each graph query.* After *every* iteration of gradient descent that changes parameters critical to the computational graph structure, e.g. the α parameters that generate traversal choices, we again query the graph database, form a new utility function with respect to those new sample paths, and perform gradient descent.

Through this approach of alternating a process of query sampling with a process of gradient descent upon the utility function defined by those samples, we allow the graph database to be fully explored. Given appropriate variables, the entire graph can be explored if required. This overall methodology is kept tractable by a combination of the SBGSE as outlined in Section 5.4, approximating the full query via sample paths, and using continuous approximations to discrete variables as outlined in Chapter 1. In the SmartGraph implementation, CGQ uses the Gorgonia [196] package for performing Stochastic Gradient Descent, a TensorFlow-like package for the Go programming language with support for dynamic computational graphs via its LispMachine.

5.7 Experiments and Results

Here we outline a series of results demonstrating the applicability of the CGQ methodology (on an Xbox Live dataset summarized below in Table 5.1). These are more general results demonstrating the implementation of the CGQ approach in the SmartGraph system. In Section 5.7.3, we formulate and show CGQ approaches specifically targeted at solving product recommendation.

Note that all our results reference α values, where we assume (unlike Chapter 1), that the α values are normalized onto the simplex. To ensure this always stays the case and does not cause problems during training (recall that we take $\log \alpha_i$, and so α values can never be allowed to become negative), we perform gradient descent on unnormalized values η_k that are allowed to be

negative, and set $\alpha = softmax(\eta)$. In this manner, we can use gradient descent methods with the α parameters always interpreted as well-behaved probability values. In this manner, the experiments below demonstrating components of α that are close to 1 is a desirable behavior, as it indicates gradient descent has located a pure strategy that is suitable for the cost function associated with optimization problem. Of course, there exist many optimization problems where a mixed strategy is optimal (well recognized in game theory and discussed in Chapter 3). For illustrative purposes, the problems below are, by construction, CGQs where we expect the optimal solution to be a pure strategy.

5.7.1 Exploration of Data

The video game industry is larger than both the film and music industries combined. Recommendation systems for video games have received relatively scant academic attention, despite the uniqueness of the medium and its data. We explore recommendations that make use of interactivity, arguably the most distinguished feature of video game products. In [6], we showed that the use of implicit data that tracks user-game interactions and levels of attainment (e.g. Microsoft Xbox Achievements) has high predictive value when making recommendations. Furthermore, we argue that the characteristics of the video gaming hobby (low cost, high duration, socially relevant) make clear the necessity of personalized, individual recommendations that can incorporate social networking information. We demonstrate a solution to this problem as an application of the CGQ methodology, and also use this data in demonstrating functionality of the CGQ system in general.

The Xbox Live marketplace and video game service includes a wide range of data services, including data relating to games, game ownership, friendships, group (e.g. gaming "clans" or social groups representing real-world groups) membership, sales data, and achievements. The Xbox Live service does not have an official publicly accessible API, and so to crawl this dataset, we used an unofficial API service [197]. In Table 5.1 and Figure 5.6, we briefly describe and summarize the graph dataset formulated from crawling the API (see [6] for a more detailed exploration of the dataset).

Table 5.1: Vertex and edge type counts in the crawled datasets

Name	Type	Description	Xbox Live Count
V_P	Vertex	Players	102861
V_G	Vertex	Games	1681
V_D	Vertex	Game developers	994
V_R	Vertex	Game genres	21
E_F	Edge	Friendships between players	12834391
E_O	Edge	Ownership of games by players	3755497
E_D	Edge	Games developed by developers	1736
E_R	Edge	Games belonging to genres	2480

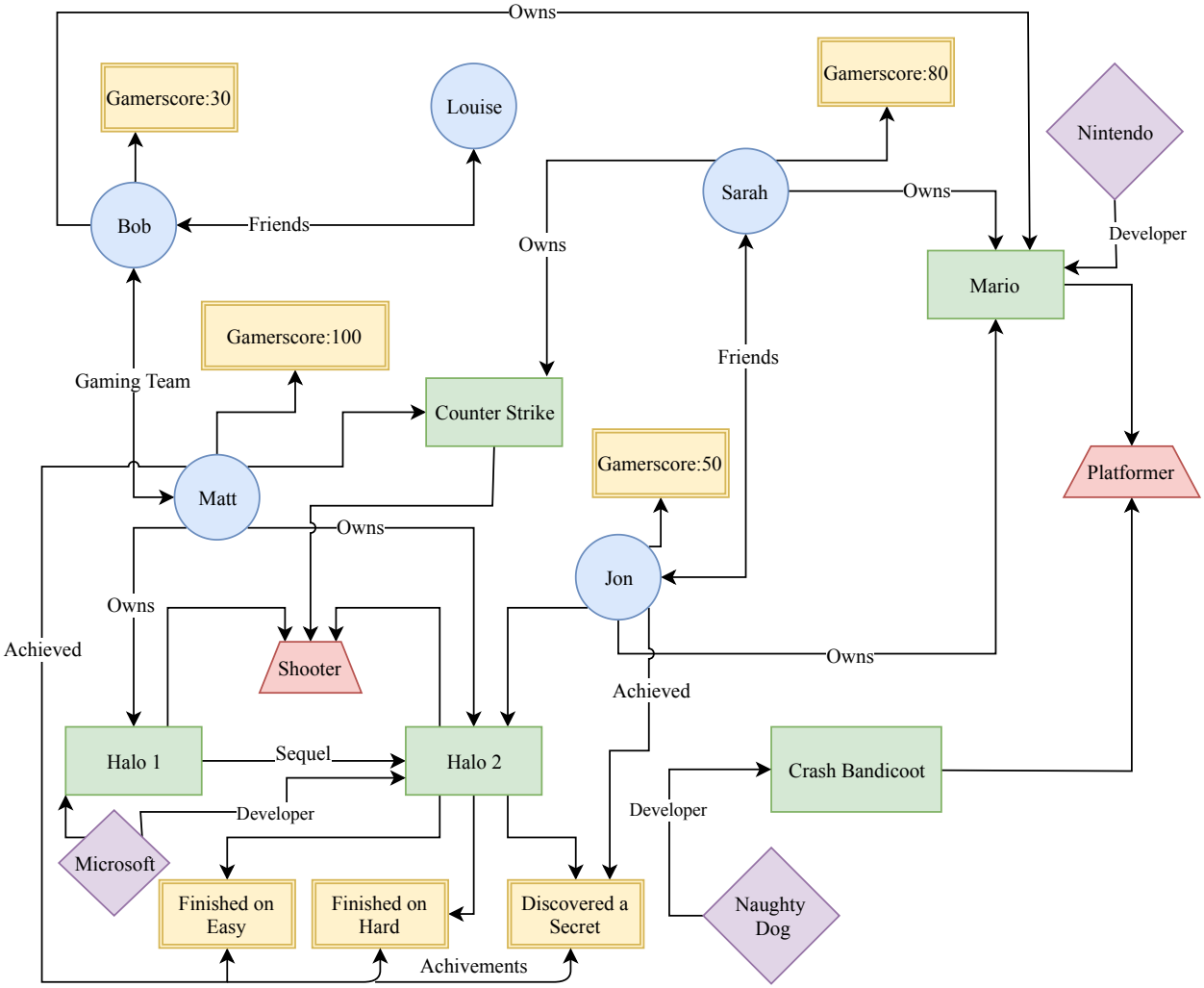


Figure 5.6: Fictional subgraph of the Xbox Live dataset, showing various node and edge types. Different vertex types are represented by different box colors and shapes, with edge types described on the edge. Properties are represented as double-lined orange rectangles, though in the SmartGraph they are stored inside the corresponding vertex or edge. We show a superset of node and edge types, e.g. the "sequel" relationship is in the full dataset, but not among the edge types considered in our experiments.

5.7.2 Learning and Convergence in the Computational Graph Query

Starting each iteration with 500 Players V_P (chosen uniformly at random in each iteration), in Figure 5.7 and Figure 5.8, we demonstrate the application of the CGQ methodology to a query where we set α parameters to generate game ownership edge samples E_O for a variety of different genres, i.e. $[\alpha_{\text{Family}}, \alpha_{\text{Social}}, \alpha_{\text{Fighting}}, \alpha_{\text{Strategy}}, \alpha_{\text{Shooter}}]$ using a split CentralOperation, as described in Section 5.1.1. The goal of the Computational Graph Query in this experiment is to "find paths from Players to Games that maximize the average Gamerscore of the players" (Gamerscore is defined in Section 5.7.3below in more precise detail, but it is essentially a measure of overall accomplishment within all the games a user owns), we use gradient descent on the relevant α parameters to select the genre of the games that the CGQ traverses such that we maximize the Gamerscore of Players. We chose this application to demonstrate how the utility function of a CGQ can depend on earlier stages of the traversal, and not just the final stage. Here, the query goes from Player nodes to Game nodes, with the utility using properties of the first Player vertex set. However, the Game nodes are critical, as they guide the path of the CGQ, because players who play particular genres tend to have higher Gamerscores (indeed, this is what the result of the CGQ tells us).

We observe some very interesting properties in Figure 5.7 and Figure 5.8. The most apparent is that Figure 5.7 appears as complete noise, whereas Figure 5.8 eventually demonstrates convergent behavior for the parameters. The noisy appearance of Figure 5.7 is to be expected, given how Section 5.6 outlines that every cost function value is actually calculated from a different set of N traversals. However, when we use the gradients of this noisy appearing cost function, the CGQ is still able to learn the correct parameter values, demonstrating that owners of games with the "Shooting" genre tend to have higher Gamerscores. This is akin to how the Player "Matt" in Figure 5.6 owns many shooting games, and has the highest Gamerscore property. These figures demonstrate an important aspect of CGQ that must be considered when using it to perform deep learning on graphs; the cost function can be used for guiding parameters in the descent, but typically can't be used for ascertaining convergence.

Figure 5.9 and Figure 5.10 investigate this phenomenon more closely, with the trivial query "which genre traversals from Players to Educational or Shooter Games have the highest average minimum age requirement". This query is trivial since the genres are restricted to "Educational" and "Shooter", where the latter often have M or R ratings, and the former no such ratings. In each

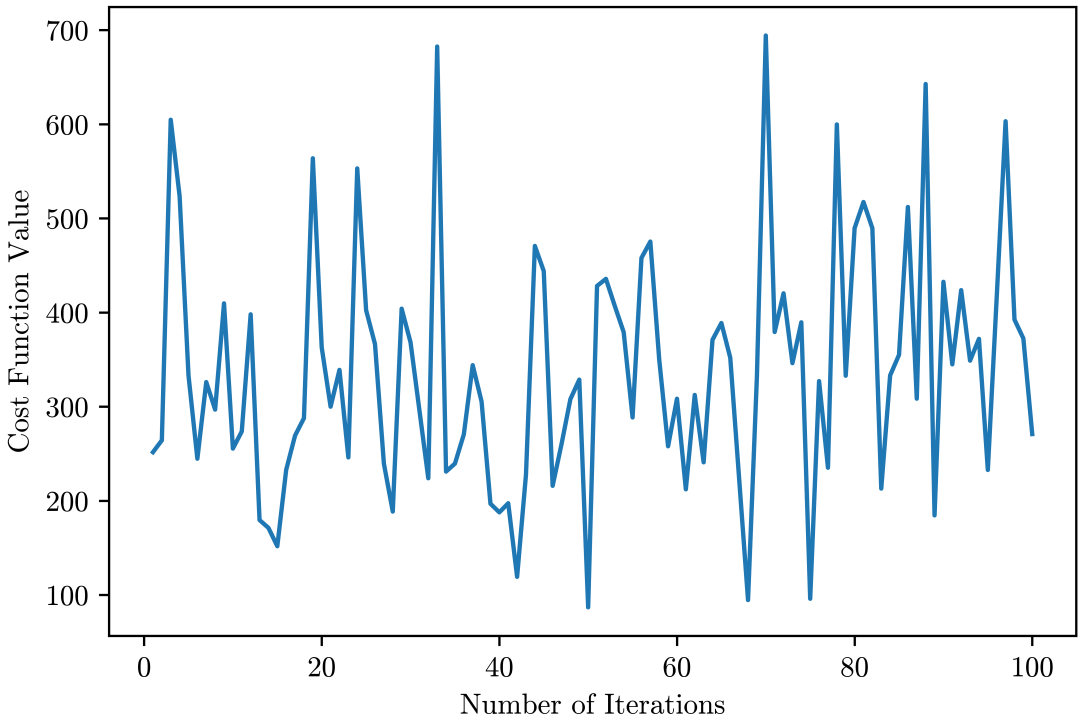


Figure 5.7: A Noisy cost function of a CGQ.

Figure, we observe the effect of increasing the number of sample paths used in each optimization step of the CGQ. While the cost function shown in Figure 5.9 is still "noisy", we observe that the magnitude of this noise reduces as we increase sample paths.

This might seem trivial (i.e. a simple result of the law of large numbers), but recall that parameter optimization and graph querying is constantly occurring during this process. Of interest is that regardless of the sample size, we observe in Figure 5.10 that all queries are quickly converging to the correct answer. This is a very important point to note about the CGQ methodology; we appear to converge to the correct values even when we don't seem to have "enough" samples to represent the graph structure. The reason this is important is related to the exponential blowup discussed above. That is, even for a moderate sized graph, there can be so many possible traversals through the graph that we could never hope to generate enough samples to make the approximation of (5.3) particularly accurate. Fortunately for the CGQ methodology, the gradients generated by the approximation still appear to guide the optimization to the correct solution. We propose that part of the reason for this is that the CGQ methodology is highly resistant to getting trapped in

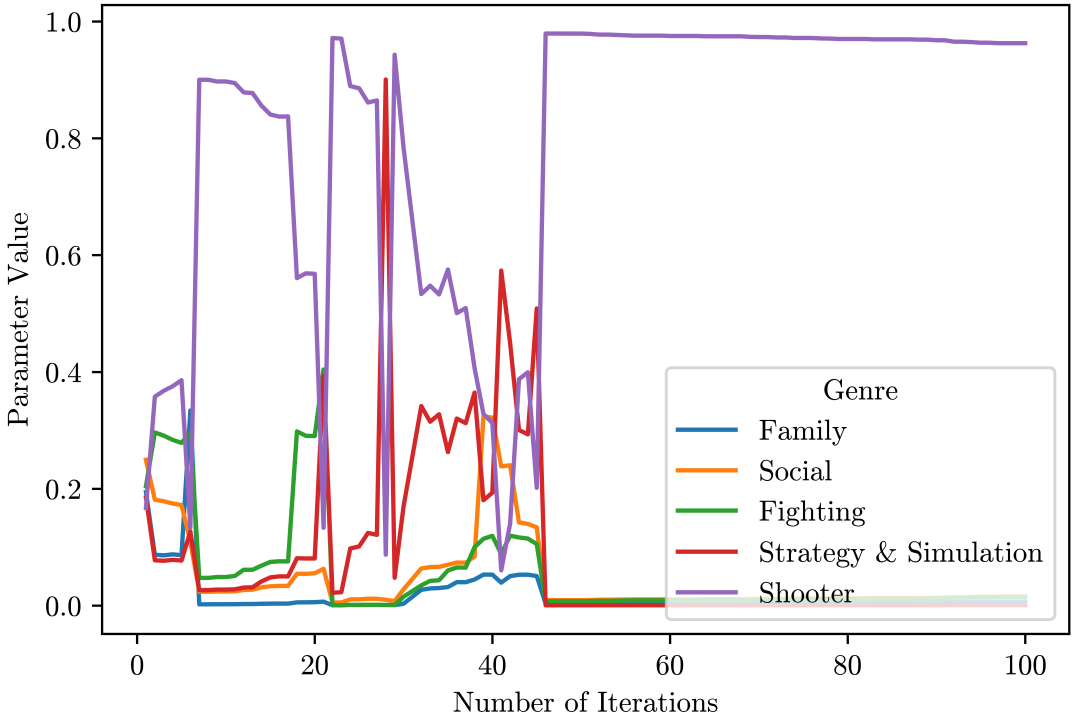


Figure 5.8: Values of α parameters associated with each genre during gradient descent on a CGQ.

local minima by virtue of the requerying process. This is because a local minima that is good for a particular set of paths is unlikely to be a good local minima for the subsequent paths generated after a new query, though this is, of course, dependent upon the graph and property structure of the database objects. The use of randomness to escape local minima is a well-studied field in optimization [198, 199]. However, there is a significant qualitative difference between the use of randomness in typical gradient descent approaches, and the randomness in CGQ. In batched SGD, randomness is introduced before function evaluation in the definition and/or choice of the data samples used in the function evaluation. In contrast, the randomness involved in CGQ comes from explorations of the graph that are potentially *the result* of the input parameters.

For very small problems, we anticipate that noise in the approximation would essentially disappear as it becomes computationally feasible to have a large enough number of samples. Indeed, we have already shown this is the case in Chapter 3. Consider the problem of performing ancestral sampling on a DeepID, with Concrete approximations to decisions used to maximize some notion of utility. Thinking of the macro-level graph structure of the DeepID quite literally, we observe

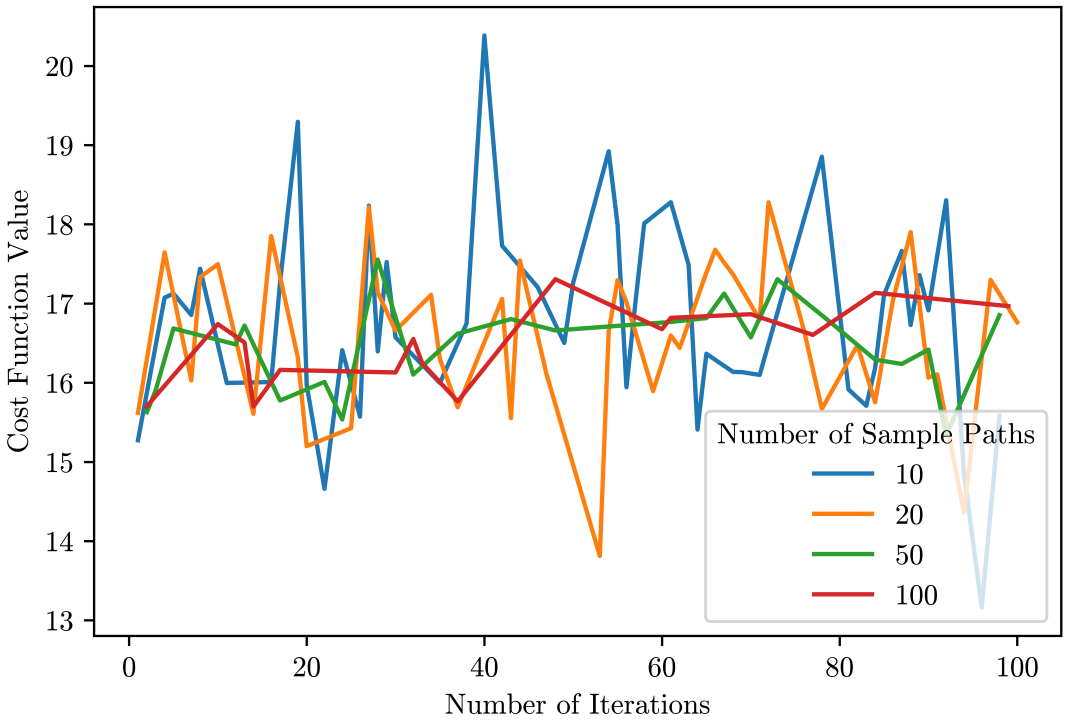


Figure 5.9: Cost function becoming more regular as samples are increased.

that DeepIDs are special cases of CGQs; they just happen to be quite small "graph databases". In the optimizations shown in Chapter 3, we observe the kind of regularity in the optimization of the cost function value that is absent from Figure 5.9. This regularity is also present in Section 5.7.4 because of the highly constrained values of the properties involved in the query.

In Figure 5.11, we demonstrate the compatibility between the CGQ methodology and the SmartGraph by showing 32 (chosen as it is double the number of threads on the machine the test was executed upon) concurrent CGQ optimizations. We use the same trivial query here, as our goal is to demonstrate that the SmartGraph can many concurrent optimizations - recall that each iteration of each optimization is also performing a large graph query that are all executing concurrently with the other graph queries and optimizations. Rather than performing simply one CGQ at a time, the SmartGraph implements the approach with the same philosophy of graph databases and graph querying, facilitating massive concurrency and multiple user usage.

In the above experiments, the SmartGraph uses three routers, all using the MAL abstraction. In Figure 5.11 we demonstrate how the facility to add more routers improves the execution time

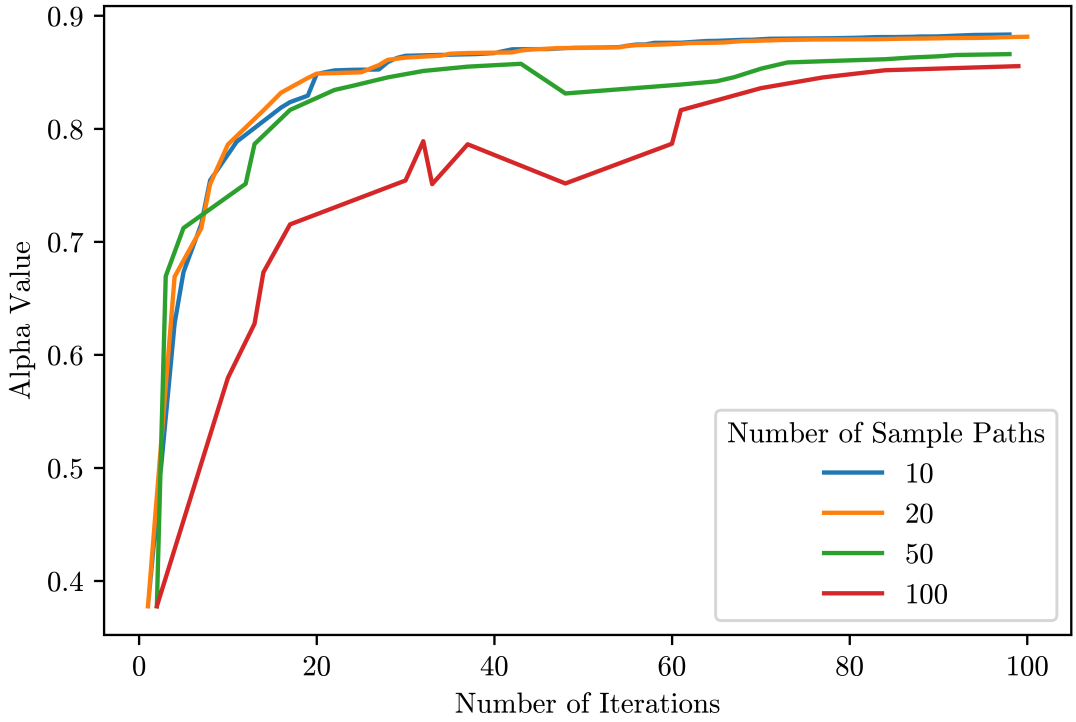


Figure 5.10: Convergent behavior in a CGQ regardless of the number of samples.

of the concurrent CGQ experiments. As we observed in Chapter 4, the optimal number of routers is not at either extreme - neither a single router, nor a router for every vertex or edge.

5.7.3 Personalized Product Recommendations using Attainment Data

In this section, we give a brief introduction to recommendation systems, and devise a new method for generating user-game scores that we generate from achievement data. Achievements are flags stored digitally within the Xbox Live dataset that indicate a player has completed a certain task (typically of some difficulty, such as finishing a game on "hard mode") within the product, that are combined (within and across owned games) to form a global "Gamerscore" for each player. Since we want scores on a Player→Game level, we construct our own method of scoring, which we call an "attainment score". We then generate recommendations using these scores as a practical example of the CGQ methodology.

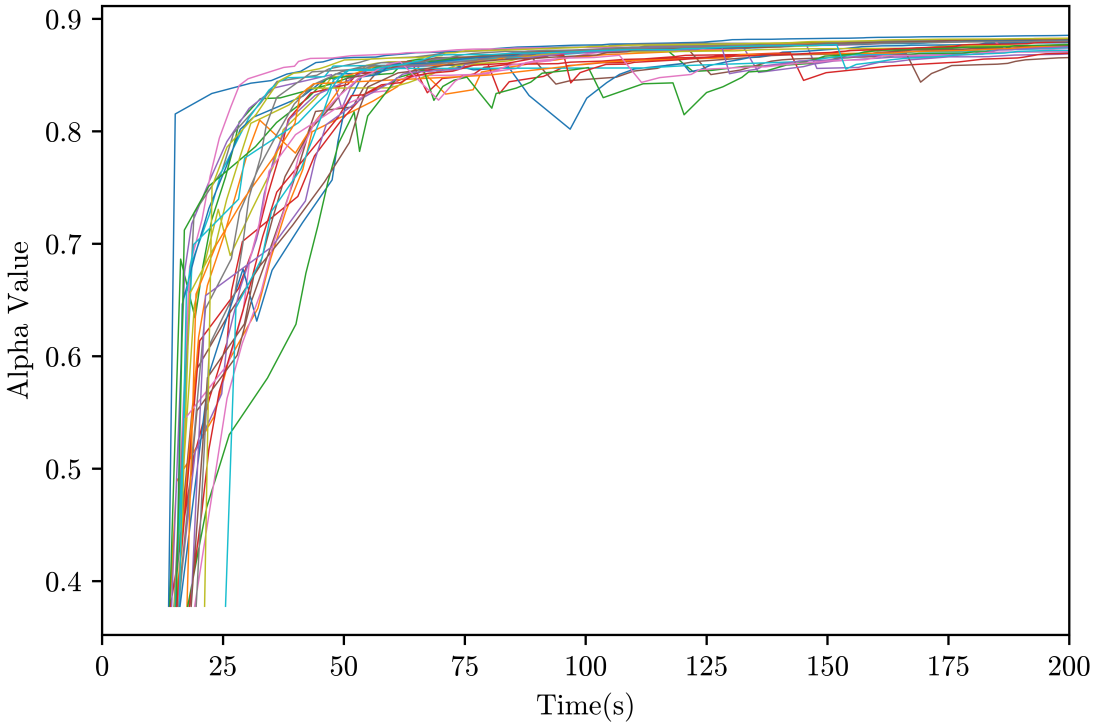


Figure 5.11: Concurrent execution of 32 CGQ optimization problems.

Introduction to Recommendation Systems and Attainment Ratings

Recommendation systems and the techniques developed for them have historically focused on music [200], film [201], and other media for which consumption is a predominantly passive process. These services (see [195] for a recent review of the field) primarily employ "explicit" ratings data, wherein a user directly and deliberately inputs some form of rating for a product. In this context, implicit data, i.e. data that may *implicitly* indicate user preferences, but that does not involve the user explicitly designating a score or writing a review, is often seen as less causally informative, more biased, and less precise. We argue that there exists a class of products for which implicit data is disproportionately relevant; interactive media products for which interaction is the primary method of consumption. Further, we show how this data can be used effectively for making recommendations. We use the video game industry and Xbox Live [202] as our source of interactive media products. Xbox Live is a digital distribution service for Windows PCs and Microsoft's Xbox line of home video game consoles. Products on Xbox Live are actively marketed to users running each

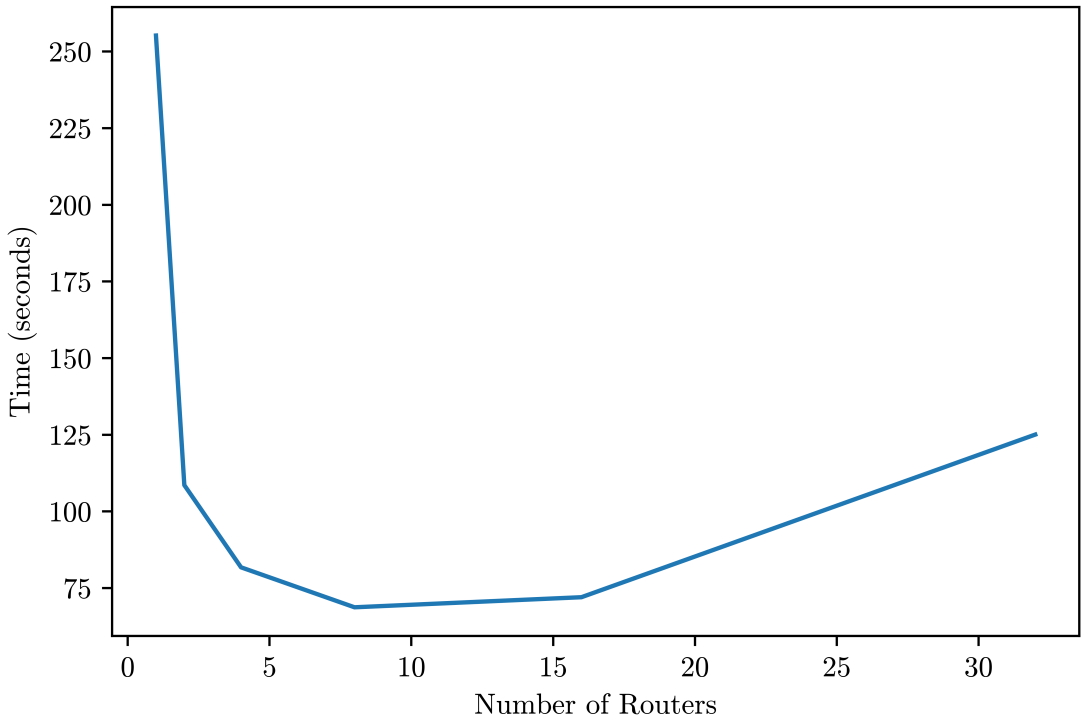


Figure 5.12: Effect on CGQ runtime as we increase the number of routers.

service, with regular discounted sales events and pop-ups of recommended purchases. Although Xbox Live does allow users to post reviews of purchased products, the use of such data in making recommendations can be called into question due to the frequency of review bombing incidents [203, 204] and the relative sparsity of the review datasets (i.e. where few users post reviews).

The need for recommendation systems capable of taking into account the particulars of interactive media products is clear. We outline a method for how this enormous implicit data set can be combined into scores between zero and one for each user-game pair and argue that the resulting score can be interpreted as an implicit rating of the game by the user with that rating having many desirable properties. Unlike many traditional implicit data sets, these ratings are not unary or binary, and demonstrate global consistency (unlike explicit rankings, where e.g. users may disagree what criteria is required for a 5-star rating).

Xbox Live is also a software platform on which purchased products are managed, organized, and *run*. It is the interactive nature of video games that distinguishes games from other forms of media. Rather than a passive process of consumption, playing video games involves making conscious

choices about how, when, and why to perform certain interactions. It is this interactive experience of the consumers with the product that they purchase (and other users who have made the same purchase) that differentiates digital distribution systems like Xbox Live from retailers like Netflix and Amazon [205] that sell physical and digital products through an online marketplace. Here we argue that the choices players make are indicative of their preference of titles, and can therefore be used for the purposes of product recommendation. Furthermore, we argue that presently recorded data, in the form of achievements, directly reflects these actions.

Query by Example in Recommendations

We detail a method for generating game recommendations by exploring the networks of Xbox Live using the CGQ methodology. The field of social recommender systems [206–209] has become increasingly important in recent years, growing alongside the size and complexity of commonly used social networks. Such work has typically focused on key domain areas such as web search [147, 210, 211] and friend recommendation in social networks [212, 213]. In its most basic form, the problem of social recommendation can be viewed as an example of link prediction on graphs, and much work has been completed from this perspective [213–215]. Indeed, many methods of recommendation in general can be viewed as such. Consider that the method of collaborative filtering [195] on user-movie-ratings triplets (e.g. of Netflix Prize ilk) can be viewed as a link prediction problem on a bipartite graph of users and movies (connected by edges denoting ratings). In social recommendation datasets, we are instead performing link prediction on a graph that is not necessarily bipartite, with many vertex (e.g. person, product, organization, news item) and edge types (friends, owns, belongs to, read). Recommendation, then, can thus be considered a graph problem.

This has led to work generating recommendations via random walks on the graph [111, 208, 216, 217]. Multi-modality in vertex and edge types has primarily been addressed through two different approaches. The first is to use an ensemble approach, where existing bipartite techniques are applied to graphs partitioned by vertex and edge type (this is required due to the tendency of a particular node or edge type to dominate when types are not considered), and recombined into overall recommendations. The second approach, more relevant to this chapter, is to bias random walks on the graph such that the biased walks are more likely to arrive at relevant products [218,

219]. These approaches work to calculate edge transition probabilities, such that the probability of any individual walk is then the product of the probabilities of the edges it contains. While these approaches are useful for finding recommendations for existing users and products, they lack explanatory power because they do not interpretably describe why or how the walks are biased. This can result in difficulty in applying supervised random walk methods to new data without additional computation.

Developed separately from supervised random walks is a related graph problem that can similarly be used to generate product recommendations. Rather than focusing on product recommendation specifically, "reverse engineering" a.k.a. "reachability problem" a.k.a. "query by example" [185–187] seeks (in the graph database context) to learn the queries on graphs that result in certain vertices (or more general graph objects) being returned (where to perform learn queries to recommend products, we set a users owned products as the example vertices). In contrast to supervised random walks, the focus of query by example is specifically on learning some approximate query within the bounds of a particular query language. Naturally, the hope is that such a query is interpretable, and that interpretation of this query is just as informative as the additional results that the learned query might yield.

Attainment Ratings

We argue that achievements are highly informative implicit data that can be used to determine a score representing how much a particular user likes a particular game. The rationale is quite simple; the more a user enjoys a game, the more they will attempt to complete everything it has to offer. We compute an overall achievement score for a particular user by combining each individual achievement weighted by difficulty level computed using global achievement rates.

Hours played has been proposed in the literature [220] as a metric to approximate user preferences. However, there are significant issues with the use of simple play times. For instance, the length of games is not uniform. Certain genres (e.g. Role-Playing games) are typically significantly longer than other genres (such as Action games), and many highly acclaimed games are extremely short (such as the Playstation title, Journey). Though we can "normalize" play times (e.g. by uniform scaling, using z-scores etc.), the process of normalization requires explicitly comparing all users (or at least, some statistically significant subsample), a computationally expensive operation.

We propose to eliminate these shortcomings by using achievement data to define what we call an attainment rating. These attainment ratings can be computed without explicitly comparing users, yet result in a score that is globally consistent across all users.

The attainment rating is computed by combining each individual achievement weighted by a difficulty level computed using global achievement rates. For a game g we denote its set of achievements as $A_g = \{A_{g_1}, A_{g_2}, ..., A_{g_{N_g}}\}$, where N_g is the total number of achievements of game g. Each $A_{g_i} \in A_g$ is a binary vector of length $\|P_g\|$, where P_g is the set of players who own game g, such that $A_{g_i s}$ indicates whether or not player s has achieved achievement number i in game g. Clearly, the proportion of players who have achieved A_{g_i} can be calculated as follows:

$$C_{g_i} = \frac{\sum_{p \in P_g} A_{g_i p}}{\|P_g\|} \tag{5.11}$$

We define the attainment score A_{gs} of player s for game g as follows:

$$A_{gs} = \sum_{A_{g_i} \in A_g} \frac{A_{g_i s} \cdot (1 - C_{g_i})}{N_g} \tag{5.12}$$

The attainment score A_{gs} has a number of desirable properties. First, we observe that for each player s, the impact of all other players on the attainment score of player s for game g is expressed solely through global achievement proportions as determined by (5.11). However, due to the gamification of obtaining and displaying achievements, the evaluations of (5.11) for all achievements are available apriori in the dataset. From (5.12) it follows that that A_{gs} aggregates a global understanding of achievement difficulty without the needing for explicit user comparisons – a rare achievement, i.e. one with $C_{g_i} \approx 0$, makes a large contribution to A_{gs}. Similarly, extremely common achievements, i.e. those with $C_{g_i} \approx 1$, contribute very little to the attainment rating. From (5.12) we have:

$$0 \leq A_{gs} \leq 1 - \frac{1}{\|P_g\|} \tag{5.13}$$

The attainment rating $A_{gs} = 0$ if the user has no achievements for game g, or in the very unlikely event that everyone else has all the achievements for it. The maximum value $A_{gs} = 1 - \frac{1}{\|P_g\|}$ is achieved in the unlikely event that user s has all achievements for game g, and is the only user to have all the achievements. In a traditional rating system, different users may have entirely

different standards for what constitutes a given score (e.g. giving out full marks for "I liked this" versus full marks for "this is literally my favorite game of all time"). In contrast, the attainment ratings in (5.12) are precise, and measure the same degree of attainment for every user. The use of achievements can give scores across the full range for all users, with truly high attainment ratings possible, but increasingly rare.

5.7.4 Computational Graph Query Recommendations

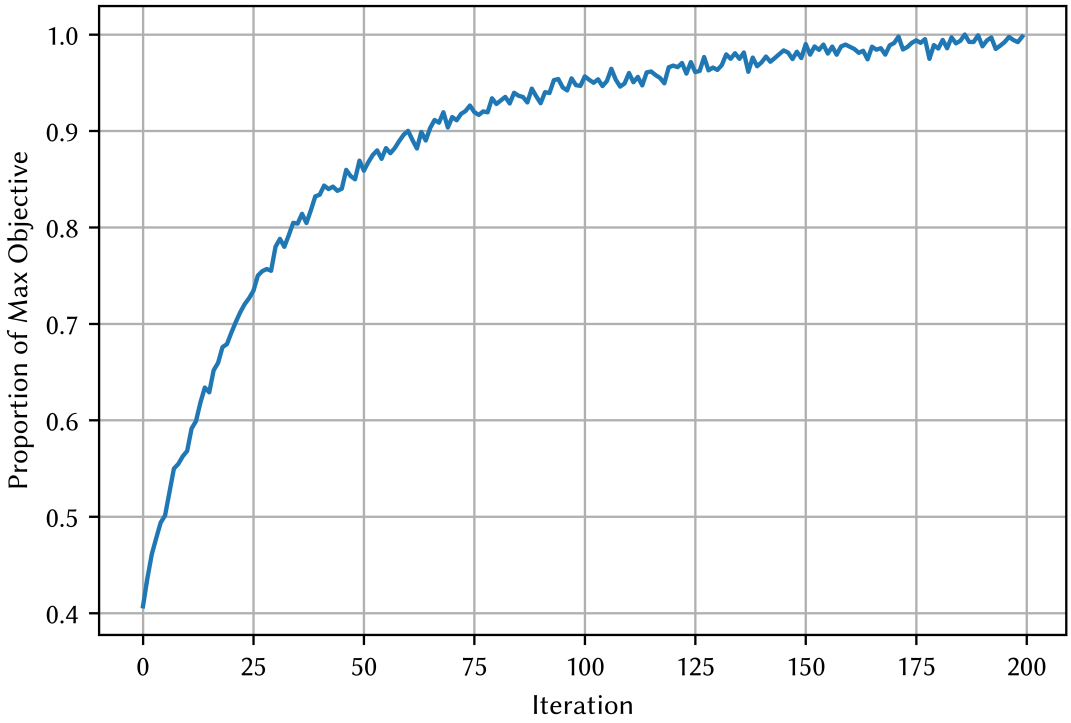


Figure 5.13: Training of a product recommendation CGQ.

The average playtime of a video game is considerably longer than many other forms of entertainment, with games in the Action genre having playtimes typically in the 10-20 hour range, and Role-Playing games often extend to *hundreds* of hours. Given the comparatively long length of video games and the large number of products in the dataset, the ability to recall even a small fraction of highly relevant products using attainment ratings suggests an ability to generate a sufficiently large number of recommendations for users for the duration of their use of the platform.

This encourages the repeated use of more personalized queries, where a user can narrow these general recommendations to match their specific desires of the moment.

The social aspect of using (attainment) rating data from actual friends (as opposed to other users designated as "similar" via some clustering or similarity measure) makes particular sense in the context of video games, where online multiplayer is often a big part of a product's appeal. These properties suggest the value of graph-based recommendations using CGQ in this setting, where users can easily specify queries that account for desired properties and social networks. We search in the graph for specific paths of nodes and edges that can succinctly and interpretably describe "complex" queries that actually represent intuitive ideas.

In query by example for recommendation systems, we might seek to learn an interpretable graph query from data. Or, we might understand aspects of the query (such as desirable product properties), but not know how best to traverse the graph to get the most relevant recommendations. To demonstrate the applicability of the CGQ approach, we parameterize a family of queries where α_{ijm} is the probability of a walk traversing from a node of type i to a node of type j on the kth traversal step s.t. $\sum_{j=1}^{K} \alpha_{ijm} = 1$ (i.e. for every step and vertex label type, there is a distribution over the vertex labels of the next step). Once a future label is chosen for each sample path according to α using the Concrete distribution, the sample path walks along an edge chosen uniformly at random from all edges that will get from the current location to the chosen vertex label. As a utility function, we use the sum of average attainment rating (note that attainment scores are not graph data here, but a pre-calculated property value for each Game vertex) for each game traversed during each walk, and sum over all walks (note the similarity to finding the average of sums of weights on paths in Figure 5.2). That is, we maximize the following objective:

$$\sum_{n=1}^{N} \sum_{m=1}^{M} x_{nm}.avgAttainment, \tag{5.14}$$

where $x_{nm}.avgAttainment$ is the average attainment (across Players who own the game) of the node arrived at from the mth traversal step of the nth sample path (where $x_{nm}.avgAttainment$ is non-zero only if the node is a game node). In Figure 5.13, we observe that simple SGD very quickly learns to optimize the CGQ. In particular, it learns the globally optimal (and interpretable) solution of "traverse repeatedly from users to owned games, to other users who own that game, to games that they own". Something of note in Figure 5.13 is that the cost function value is relatively stable,

unlike the examples in Figure 5.7 and Figure 5.9. We suggest this is because of the stability of problem 5.14; with N and M fixed, and with attainment ratings continuous in the interval $[0, 1]$, it is much less likely that subsequent iterations will have huge deviations, even as they sample different paths. This is in contrast to the queries examined previously, which depended on parameters like Gamerscore that are unbounded, and minimum age requirements, which are categorical. To aid with interpretability, Figure 5.13 is shown with respect to the maximum objective, which was found by manually enumerating each combination of traversal sequences (something that cannot be done with more complicated queries, as discussed in Section 5.4).

5.8 Discussion

The Computational Graph Query methodology described in this chapter is a potential solution to a problem that has plagued machine learning researchers since deep learning first began to gain serious traction. Instead of approximating graphs as tensors, CGQ allows us to retain all structural and property information contained in a graph database. Furthermore, it allows the machine learning algorithms to interpret graph structured data as an actual graph, through inspection in the same way that users interpret large graph structures, that is, by querying. There is a cost to pay, of course, in potentially having to generate numerous traversals. Fortunately, all of our results indicate that in practice, we can solve problems of note with only a moderate number of traversals.

Through both simple examples and the important use-case of recommendation systems, we outlined how CGQ exhibits a lot of promise, and performs well even in environments where we cannot hope to obtain "enough" path samples to fully approximate a graph. Importantly, we demonstrated how CGQ can work in a real graph computing environment, the SmartGraph as outlined in Chapter 4. Instead of merely getting CGQ running in the SmartGraph, we used the concurrent capabilities of the system to demonstrate how CGQ could be used to solve graph based machine learning problems for many users simultaneously.

Conclusion

In this dissertation, we outlined how aspects of three typically unrelated fields could be combined synergistically to solve outstanding problems in each field.

In deep learning, we demonstrated how the concept of graph querying, as it has been developed in graph databases, can be used to solve the problem of inputting highly variable graph structured data into machine learning models without hammering it to fit the existing tensor-focused environment of traditional deep learning systems. This method of interpreting graph structured data in a more natural, interpretable way is one of the most significant accomplishments of this dissertation, and also indicates many possible directions for future research (as there are many graph-input based problems to which this methodology could be applied).

In graph computing, we demonstrated how the concurrent execution models developed in graph databases can be used to execute complex analytics. For graph analysis in general, we showed how continuous approximations for discrete systems allow us to incorporate a wide range of deep learning techniques (such as Stochastic Gradient Descent) into situations where it is typically not possible to do so.

We further demonstrated the importance of having interpretability in the models that we use. Not only did it allow us to easily make robust modifications to the DeepID decision support system (which we later revealed as a special case of a Computational Graph Query), it was also a key concept behind the development of the Computational Graph Query methodology; interpretability leads to "natural understanding", and thereby natural language processing. For not only is interpretability important to human users of algorithms and analytics (e.g. in understanding the how and why of generated results), it is also important to the execution of the algorithms themselves. By giving machine learning models the ability to interpret graph structured datsets in the

same way as humans (i.e. interpretable path traversals), we facilitated important new solution methodologies.

As a systems dissertation, we demonstrated how many of the theoretical and abstract ideas of this dissertation could be implemented in a real-world combined graph database, graph computing, machine learning system. This allowed us to explore and understand how many of these ideas actually mesh and can be applied in practice. While some combinations seemed atypical (e.g. implementing CGQ on multiple routers), we ultimately observed that these combinations were beneficial, either resulting in increased performance, or entirely new capabilities. It is our hope that the ideas and philosophy behind this dissertation encourages others to examine how traditionally disparate fields can be used both to generate new academic research, and to build practical industrial systems.

Bibliography

[1] S. Linderman, G. Mena, H. Cooper, L. Paninski, and J. Cunningham, "Reparameterizing the Birkhoff Polytope for Variational Permutation Inference," in *International Conference on Artificial Intelligence and Statistics*, 2018, pp. 1618–1627.

[2] W. Liu, H. Cooper, M.-h. Oh, P.-Y. Chen, S. Yeung, F. Yu, and T. Suzumura, "Learning Graph Topological Features via GAN," *IEEE Access*, vol. 7, pp. 21 834–21 843, 2019.

[3] H. Cooper, G. Iyengar, and C.-Y. Lin, "Deep Influence Diagrams: An Interpretable and Robust Decision Support System," in *22nd International Conference on Business Information Systems*, Seville, 2019.

[4] ——, "Interpretable Robust Decision Making," in *International Conference on Autonomous Agents and Multiagent Systems*, Stockholm, 2018.

[5] ——, "Smartgraph: An Artificially Intelligent Graph Database," in *7th International Conference of Advanced Computer Science & Information Technology (ACSIT 2019)*, Sydney: AIRCC Publishing Corporation, 2019, pp. 63–77. DOI: 10.5121/csit.2019.90307.

[6] ——, "Personalized Product Recommendation for Interactive Media," in *International Conference on Intelligent Human Systems Integration*, Springer, 2019, pp. 510–516.

[7] D. M. Blei, A. Kucukelbir, and J. D. McAuliffe, "Variational inference: A review for statisticians," *Journal of the American Statistical Association*, 2017.

[8] T. Salimans and D. A. Knowles, "Fixed-Form Variational Posterior Approximation through Stochastic Linear Regression," *Bayesian Analysis*, vol. 8, no. 4, pp. 837–882, 2013.

[9] D. P. Kingma and M. Welling, "Stochastic Gradient VB and the Variational Auto-Encoder," *2nd International Conference on Learning Representationsm (ICLR)*, pp. 1–14, 2014, ISSN: 0004-6361. DOI: 10.1051/0004-6361/201527329. arXiv: 1312.6114. [Online]. Available: http://arxiv.org/abs/1312.6114.

[10] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, no. 3–4, pp. 229–256, 1992.

[11] P. W. Glynn, "Likelihood ratio gradient estimation for stochastic systems," *Communications of the ACM*, vol. 33, no. 10, pp. 75–84, Oct. 1990.

[12] C. J. Maddison, A. Mnih, and Y. W. Teh, "The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables," *Proceedings of the International Conference on Learning Representations*, 2016.

[13] E. Jang, S. Gu, and B. Poole, "Categorical Reparameterization with Gumbel-Softmax," *Proceedings of the International Conference on Learning Representations*, 2016.

[14] R. Williams, "A class of gradient-estimation algorithms for reinforcement learning in neural networks," in *Proceedings of the International Conference on Neural Networks*, 1987, pp. II–601.

[15] W. Grathwohl, D. Choi, Y. Wu, G. Roeder, and D. Duvenaud, "Backpropagation Through the Void : Optimizing Control Variates for Black-Box Gradient Estimation," *ICLR*, no. 2017, pp. 1–15, 2018. [Online]. Available: `https://openreview.net/pdf?id=SyzKd1bCW`.

[16] E. J. Gumbel, *Statistical theory of extreme values and some practical applications: a series of lectures*, 33. US Govt. Print. Office, 1954.

[17] G. Tucker, A. Mnih, C. J. Maddison, D. Lawson, and J. Sohl-Dickstein, "REBAR: Low-variance, unbiased gradient estimates for discrete latent variable models," no. Nips, 2017. arXiv: `1703.07370`. [Online]. Available: `http://arxiv.org/abs/1703.07370`.

[18] T. C. Hesterberg and B. L. Nelson, "Control variates for probability and quantile estimation," *Management Science*, 1998, ISSN: 00251909. DOI: `10.1287/mnsc.44.9.1295`.

[19] L. J. Guibas, "The identity management problem: a short survey," in *11th International Conference on Information Fusion*, IEEE, 2008, pp. 1–7.

[20] J. Shin, N. Lee, S. Thrun, and L. Guibas, "Lazy inference on object identities in wireless sensor networks," in *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks*, IEEE Press, 2005, p. 23.

[21] R. Kondor, A. Howard, and T. Jebara, "Multi-object tracking with representations of the symmetric group.," in *Artificial Intelligence and Statistics*, 2007, p. 5.

[22] M. Meilua, K. Phadnis, A. Patterson, and J. Bilmes, "Consensus ranking under the exponential model," in *Proceedings of the 23rd Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, 2007.

[23] G. Lebanon and Y. Mao, "Non-parametric modeling of partially ranked data," *Journal of Machine Learning Research*, vol. 9, no. Oct, pp. 2401–2429, 2008.

[24] R. P. Adams and R. S. Zemel, "Ranking via Sinkhorn Propagation," *arXiv*, 2011. arXiv: `1106.1925`.

[25] B. Bloem-Reddy and P. Orbanz, "Random Walk Models of Network Formation and Sequential Monte Carlo Methods for Graphs," *arXiv preprint arXiv:1612.06404*, 2016.

[26] V. Rao, R. P. Adams, and D. D. Dunson, "Bayesian inference for Matérn repulsive processes," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 2016.

[27] S. Kato, H. S. Kaplan, T. Schrödel, S. Skora, T. H. Lindsay, E. Yemini, S. Lockery, and M. Zimmer, "Global Brain Dynamics Embed the Motor Command Sequence of {C}aenorhabditis elegans," *Cell*, vol. 163, no. 3, pp. 656–669, 2015, ISSN: 0092-8674.

[28] J. P. Nguyen, F. B. Shipley, A. N. Linder, G. S. Plummer, M. Liu, S. U. Setru, J. W. Shaevitz, and A. M. Leifer, "Whole-brain calcium imaging with cellular resolution in freely behaving Caenorhabditis elegans," *Proceedings of the National Academy of Sciences*, vol. 113, no. 8, E1074–E1081, 2016.

[29] J. G. White, E. Southgate, J. N. Thomson, and S. Brenner, "The structure of the nervous system of the nematode Caenorhabditis elegans: the mind of a worm," *Phil. Trans. R. Soc. Lond*, vol. 314, pp. 1–340, 1986.

[30] H. W. Kuhn, "The Hungarian method for the assignment problem," *Naval research logistics quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.

[31] J. Munkres, "Algorithms for the assignment and transportation problems," *Journal of the Society for Industrial and Applied Mathematics*, vol. 5, no. 1, pp. 32–38, 1957.

[32] E. L. Lawler, "The quadratic assignment problem," *Management science*, vol. 9, no. 4, pp. 586–599, 1963.

[33] P. Diaconis, "Group representations in probability and statistics," in *Institute of Mathematical Statistics Lecture Notes—Monograph Series*, S. S. Gupta, Ed., vol. 11, 1988.

[34] J. W. Miller and M. T. Harrison, "Exact sampling and counting for fixed-margin matrices," *The Annals of Statistics*, vol. 41, no. 3, pp. 1569–1592, 2013.

[35] M. T. Harrison and J. W. Miller, "Importance sampling for weighted binary random matrices with specified margins," *arXiv preprint arXiv:1301.3928*, 2013.

[36] J. Huang, C. Guestrin, and L. Guibas, "Fourier theoretic probabilistic inference over permutations," *Journal of Machine Learning Research*, vol. 10, no. May, pp. 997–1070, 2009.

[37] C. H. Lim and S. Wright, "Beyond the Birkhoff polytope: Convex relaxations for vector permutation problems," in *Advances in Neural Information Processing Systems*, 2014, pp. 2168–2176.

[38] S. M. Plis, S. McCracken, T. Lane, and V. D. Calhoun, "Directional Statistics on Permutations," in *Artificial Intelligence and Statistics*, 2011, pp. 600–608.

[39] M. D. Hoffman, D. M. Blei, C. Wang, and J. Paisley, "Stochastic variational inference," *The Journal of Machine Learning Research*, vol. 14, no. 1, pp. 1303–1347, 2013.

[40] D. J. Rezende, S. Mohamed, and D. Wierstra, "Stochastic Backpropagation and Approximate Inference in Deep Generative Models," *Proceedings of The 31st International Conference on Machine Learning*, pp. 1278–1286, 2014. arXiv: 1401.4082. [Online]. Available: http://arxiv.org/abs/1401.4082.

[41] R. Sinkhorn and P. Knopp, "Concerning nonnegative matrices and doubly stochastic matrices," *Pacific Journal of Mathematics*, vol. 21, no. 2, pp. 343–348, 1967.

[42] K. Li, K. Swersky, and R. Zemel, "Efficient Feature Learning using Perturb-and-MAP," *Neural Information Processing Systems Workshop on Perturbations, Optimization, and Statistics*, 2013.

[43] J. Guiver and E. Snelson, "Bayesian inference for Plackett-Luce ranking models," in *proceedings of the 26th annual international conference on machine learning*, ACM, 2009, pp. 377–384.

[44] J. Sethuraman, "A constructive definition of Dirichlet priors," *Statistica sinica*, pp. 639–650, 1994.

[45] C. L. Mallows, "Non-null ranking models. I," *Biometrika*, vol. 44, no. 1/2, pp. 114–130, 1957.

[46] L. R. Varshney, B. L. Chen, E. Paniagua, D. H. Hall, and D. B. Chklovskii, "Structural properties of the Caenorhabditis elegans neuronal network," *PLoS Computational Biology*, vol. 7, no. 2, e1001066, 2011.

[47] R. Lints, Z. F. Altun, H. Weng, T. Stephney, G. Stephney, M. Volaski, and D. H. Hall, "WormAtlas Update," in *International Worm Meeting*, 2005.

[48] P. Diaconis, "The Markov chain Monte Carlo revolution," *Bulletin of the American Mathematical Society*, vol. 46, no. 2, pp. 179–205, 2009.

[49] J. T. Vogelstein, J. M. Conroy, V. Lyzinski, L. J. Podrazik, S. G. Kratzer, E. T. Harley, D. E. Fishkind, R. J. Vogelstein, and C. E. Priebe, "Fast approximate quadratic programming for graph matching," *PLOS one*, vol. 10, no. 4, e0121002, 2015.

[50] D. Maclaurin, D. Duvenaud, and R. P. Adams, "Autograd: Effortless gradients in numpy," in *ICML 2015 AutoML Workshop*, 2015.

[51] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[52] S. Linderman, M. Johnson, and R. P. Adams, "Dependent Multinomial Models Made Easy: Stick-Breaking with the Polya-gamma Augmentation," in *Advances in Neural Information Processing Systems*, 2015, pp. 3456–3464.

[53] C. Naesseth, F. Ruiz, S. Linderman, and D. Blei, "Reparameterization Gradients through Acceptance-Rejection Sampling Algorithms," in *Artificial Intelligence and Statistics*, 2017, pp. 489–498.

[54] P. Kumaraswamy, "A generalized probability density function for double-bounded random processes," *Journal of Hydrology*, vol. 46, no. 1-2, pp. 79–88, 1980.

[55] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, X. Zheng, and G. Brain, "TensorFlow: A System for Large-Scale Machine Learning," *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16)*, pp. 265–284, 2016. arXiv: 1605.08695.

[56] W. Liu, P.-Y. Chen, H. Cooper, M.-h. Oh, S. Yeung, and T. Suzumura, "Can GAN Learn Topological Features of a Graph?" In *ICML 2017 Workshop on Implicit Models*, 2017.

[57] S. Muppidi and V. N. Koraganji, "Survey of contemporary ranking algorithms," *International Journal of Applied Engineering Research*, vol. 11, no. 1, pp. 322–325, 2016.

[58] S. Fortunato, "Community detection in graphs," *Physics Reports*, vol. 486, no. 3, pp. 75–174, 2010.

[59] V. Martinez, F. Berzal, and J.-C. Cubero, "A Survey of Link Prediction in Complex Networks," *ACM Computing Surveys (CSUR)*, vol. 49, no. 4, p. 69, 2016, ISSN: 03600300. DOI: 10.1145/3012704.

[60] Y. Wu, N. Cao, D. Archambault, Q. Shen, H. Qu, and W. Cui, "Evaluation of Graph Sampling: A Visualization Perspective," *IEEE Transactions on Visualization and Computer Graphics*, vol. 23, no. 1, pp. 401–410, 2017.

[61] J. Xiang, T. Hu, Y. Zhang, K. Hu, J.-M. Li, X.-K. Xu, C.-C. Liu, and S. Chen, "Local modularity for community detection in complex networks," *Physica A: Statistical Mechanics and its Applications*, vol. 443, pp. 451–459, 2016.

[62] A. Delis, A. Ntoulas, and P. Liakos, "Scalable link community detection: A local dispersion-aware approach," in *Big Data (Big Data), 2016 IEEE International Conference on*, IEEE, 2016, pp. 716–725.

[63] J. Leskovec and C. Faloutsos, "Sampling from large graphs," in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, 2006, pp. 631–636.

[64] J. Leskovec, J. Kleinberg, and C. Faloutsos, "Graphs over time: densification laws, shrinking diameters and possible explanations," in *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, ACM, 2005, pp. 177–187.

[65] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world'networks," *nature*, vol. 393, no. 6684, pp. 440–442, 1998.

[66] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *science*, vol. 286, no. 5439, pp. 509–512, 1999.

[67] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *arXiv preprint arXiv:1511.06434*, 2015.

[68] D. J. Watts, P. S. Dodds, and M. E. J. Newman, "Identity and search in social networks," *science*, vol. 296, no. 5571, pp. 1302–1305, 2002.

[69] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," *Advances in neural information processing systems*, pp. 2672–2680, 2014.

[70] I. Goodfellow, "NIPS 2016 Tutorial: Generative Adversarial Networks," *arXiv preprint arXiv:1701.00160*, 2016.

[71] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of statistical mechanics: theory and experiment*, vol. 2008, no. 10, P10008, 2008.

[72] G. Karypis and V. Kumar, "METIS - unstructured graph partitioning and sparse matrix ordering system, version 2.0," 1995.

[73] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, "Gated Graph Sequence Neural Networks," *ICLR*, no. 1, pp. 1–19, 2015, ISSN: 0031-9007. DOI: 10.1103/PhysRevLett.116.082003. arXiv: 1511.05493. [Online]. Available: http://arxiv.org/abs/1511.05493.

[74] J. Bruna, W. Zaremba, A. Szlam, and Y. Lecun, "Spectral networks and locally connected networks on graphs," in *International Conference on Learning Representations (ICLR2014), CBLS, April 2014*, 2014.

[75] M. Henaff, J. Bruna, and Y. LeCun, "Deep convolutional networks on graph-structured data," *arXiv preprint arXiv:1506.05163*, 2015.

[76] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams, "Convolutional networks on graphs for learning molecular fingerprints," in *Advances in neural information processing systems*, 2015, pp. 2224–2232.

[77] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Advances in Neural Information Processing Systems*, 2016, pp. 3837–3845.

[78] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[79] S. Tavakoli, A. Hajibagheri, and G. Sukthankar, "Learning Social Graph Topologies using Generative Adversarial Neural Networks," 2017.

[80] E. L. Denton, S. Chintala, R. Fergus, *et al.*, "Deep Generative Image Models using a Laplacian Pyramid of Adversarial Networks," in *Advances in neural information processing systems*, 2015, pp. 1486–1494.

[81] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel, "InfoGAN: interpretable representation learning by information maximizing Generative Adversarial Nets," in *Neural Information Processing Systems (NIPS)*, 2016.

[82] J. Zhao, M. Mathieu, and Y. LeCun, "Energy-based generative adversarial network," *arXiv preprint arXiv:1609.03126*, 2016.

[83] S. Nowozin, B. Cseke, and R. Tomioka, "f-GAN: Training generative neural samplers using variational divergence minimization," in *Advances in Neural Information Processing Systems*, 2016, pp. 271–279.

[84] W. R. Tan, C. S. Chan, H. Aguirre, and K. Tanaka, "ArtGAN: Artwork Synthesis with Conditional Categorial GANs," *arXiv preprint arXiv:1702.03410*, 2017.

[85] T. Miyato, A. M. Dai, and I. Goodfellow, "Adversarial Training Methods for Semi-Supervised Text Classification," *arXiv preprint arXiv:1605.07725*, 2016.

[86] Z. Yi, H. Zhang, P. T. Gong, *et al.*, "DualGAN: Unsupervised Dual Learning for Image-to-Image Translation," *arXiv preprint arXiv:1704.02510*, 2017.

[87] A. Kuefler, J. Morton, T. Wheeler, and M. Kochenderfer, "Imitating driver behavior with generative adversarial networks," *arXiv preprint arXiv:1701.06699*, 2017.

[88] S. Kohl, D. Bonekamp, H.-P. Schlemmer, K. Yaqubi, M. Hohenfellner, B. Hadaschik, J.-P. Radtke, and K. Maier-Hein, "Adversarial Networks for the Detection of Aggressive Prostate Cancer," *arXiv preprint arXiv:1702.08014*, 2017.

[89] Wikipedia. (2017). Erdős–Rényi model, [Online]. Available: `https://en.wikipedia.org/wiki/Erdos%7B%5C%%7D5C-Renyi%7B%5C_%7Dmodel`.

[90] J. Leskovec, D. Chakrabarti, J. Kleinberg, C. Faloutsos, and Z. Ghahramani, "Kronecker graphs: An approach to modeling networks," *Journal of Machine Learning Research*, vol. 11, no. Feb, pp. 985–1042, 2010.

[91] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring network structure, dynamics, and function using NetworkX," in *7th Python in Science Conference (SciPy 2008)*, 2008.

[92] SNAP. (2017). Stanford Network Analysis Project, [Online]. Available: `http://snap.stanford.edu/`.

[93] ——, (2012). Social circles: Facebook, [Online]. Available: `http://snap.stanford.edu/data/egonets-Facebook.html`.

[94] ——, (2010). Wikipedia vote network, [Online]. Available: `http://snap.stanford.edu/data/wiki-Vote.html`.

[95] ——, (2007). Gnutella peer-to-peer network, August 4 2002, [Online]. Available: `http://snap.stanford.edu/data/p2p-Gnutella04.html`.

[96] ——, (2009). Pennsylvania road network, [Online]. Available: http://snap.stanford.edu/data/roadNet-PA.html.

[97] M. E. J. Newman, "Modularity and community structure in networks," *Proceedings of the national academy of sciences*, vol. 103, no. 23, pp. 8577–8582, 2006.

[98] C. D. Meyer, *Matrix analysis and applied linear algebra*. Siam, 2000, vol. 2.

[99] V. D. Blondel, A. Gajardo, M. Heymans, P. Senellart, and P. Van Dooren, "A measure of similarity between graph vertices: Applications to synonym extraction and web searching," *SIAM review*, vol. 46, no. 4, pp. 647–666, 2004.

[100] M. Heymans and A. K. Singh, "Deriving phylogenetic trees from the similarity analysis of metabolic pathways," *Bioinformatics*, vol. 19, no. suppl_1, pp. i138–i146, 2003.

[101] X. Kong and P. S. Yu, "Multi-label feature selection for graph classification," in *Proceedings - IEEE International Conference on Data Mining, ICDM*, 2010, ISBN: 9780769542560. DOI: 10.1109/ICDM.2010.58.

[102] Y. Zhu, J. X. Yu, H. Cheng, and L. Qin, "Graph classification: A diversified discriminative feature selection approach," in *ACM International Conference Proceeding Series*, 2012, ISBN: 9781450311564. DOI: 10.1145/2396761.2396791.

[103] X. Kong and P. S. Yu, "Semi-supervised feature selection for graph classification," in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2010, ISBN: 9781450300551. DOI: 10.1145/1835804.1835905.

[104] H. Tong, B. Gallagher, C. Faloutsos, and T. Eliassi-Rad, "Fast best-effort pattern matching in large attributed graphs," *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, p. 737, 2007. DOI: 10.1145/1281192.1281271. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1281192.1281271.

[105] R. Dreżewski, J. Sepielak, and W. Filipkowski, "The application of social network analysis algorithms in a system supporting money laundering detection," *Information Sciences*, vol. 295, pp. 18–32, 2015, ISSN: 0020-0255. DOI: http://dx.doi.org/10.1016/j.ins.2014.10.015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0020025514009979.

[106] A. Awasthi, "Clustering algorithms for anti-money laundering using graph theory and social network analysis," 2012.

[107] A. Grover and J. Leskovec, "Node2Vec," *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '16*, pp. 855–864, 2016, ISSN: 2154-817X. DOI: 10.1145/2939672.2939754. arXiv: 1607.00653. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2939672.2939754.

[108] H. Cai, V. W. Zheng, and K. C. C. Chang, "A Comprehensive Survey of Graph Embedding: Problems, Techniques, and Applications," *IEEE Transactions on Knowledge and Data Engineering*, 2018, ISSN: 15582191. DOI: 10.1109/TKDE.2018.2807452.

[109] Z. Xu, F. Zhu, S. Wang, and J. Huang, "Seq2seq fingerprint: An unsupervised deep molecular embedding for drug discovery," in *ACM-BCB 2017 - Proceedings of the 8th ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics*, 2017, ISBN: 9781450347228. DOI: 10.1145/3107411.3107424.

[110] S. Bonner, J. Brennan, G. Theodoropoulos, I. Kureshi, and A. S. McGough, "Deep topology classification: A new approach for massive graph classification," *Proceedings - 2016 IEEE International Conference on Big Data, Big Data 2016*, pp. 3290–3297, 2016, ISSN: 1408032X. DOI: 10.1109/BigData.2016.7840988. arXiv: arXiv:0811.2183v2. [Online]. Available: https://doi.org/10.1109/BigData.2016.7840988%20http://dro.dur.ac.uk.

[111] J. B. Lee and X. Kong, "Skip-Graph Learning Graph Embeddings With an Encoder-Decoder Model," *Iclr2017*, no. 2015, pp. 1–8, 2017. [Online]. Available: https://openreview.net/pdf?id=BkSqjHqxg.

[112] P. Goyal and E. Ferrara, "Graph Embedding Techniques, Applications, and Performance: A Survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017, ISSN: 09507051. DOI: 10.1016/j.knosys.2018.03.022. [Online]. Available: https://arxiv.org/pdf/1705.02801.pdf.

[113] A. Dimitrios, L. Helen, and T. Mike, "Determinants of non-performing loans: Evidence from Euro-area countries," *Finance Research Letters*, vol. 18, pp. 116–119, 2016, ISSN: 15446123. DOI: 10.1016/j.frl.2016.04.008. [Online]. Available: http://dx.doi.org/10.1016/j.frl.2016.04.008.

[114] R. A. Howard, *Readings on the principles and applications of decision analysis*. Strategic Decisions Group, 1983, vol. 1.

[115] C. Bielza, M. Gómez, and P. P. Shenoy, "A review of representation issues and modeling challenges with influence diagrams," *Omega*, vol. 39, no. 3, pp. 227–241, 2011, ISSN: 0305-0483.

[116] F. V. Jensen and T. D. Nielsen, "Probabilistic decision graphs for optimization under uncertainty," *Annals of Operations Research*, vol. 204, no. 1, pp. 223–248, 2013.

[117] F. J. Diez, M. Yebra, I. Bermejo, M. A. Palacios-Alonso, M. A. Calleja, M. Luque, and J. Perez-Martin, "Markov influence diagrams: a graphical tool for cost-effectiveness analysis," *Medical Decision Making*, vol. 37, no. 2, pp. 183–195, 2017.

[118] R. J. Boucherie and N. M. Van Dijk, *Markov decision processes in practice*. Springer, 2017, vol. 248.

[119] P. Magni, S. Quaglini, M. Marchetti, and G. Barosi, "Deciding when to intervene: a Markov decision process approach," *International Journal of Medical Informatics*, vol. 60, no. 3, pp. 237–253, 2000.

[120] G. N. Iyengar, "Robust dynamic programming," *Mathematics of Operations Research*, vol. 30, no. 2, pp. 257–280, 2005.

[121] A. Nilim and L. El Ghaoui, "Robust control of Markov decision processes with uncertain transition matrices," *Operations Research*, vol. 53, no. 5, pp. 780–798, 2005.

[122] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, pp. 694–696.

[123] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *stat*, vol. 1050, p. 10, 2014.

[124] M. Mirza and S. Osindero, "Conditional generative adversarial nets," *arXiv*, 2014. arXiv: 1411.1784.

[125] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, "Improved training of wasserstein GANs," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, Curran Associates Inc., 2017, pp. 5769–5779.

[126] H. Larochelle and I. Murray, "The Neural Autoregressive Distribution Estimator.," in *AISTATS*, vol. 1, 2011, p. 2.

[127] T. Everitt, P. A. Ortega, E. Barnes, and S. Legg, "Understanding Agent Incentives using Causal Influence Diagrams, Part I: Single Action Settings," *arXiv preprint arXiv:1902.09980*, 2019.

[128] M. Gomez, C. Bielza, J. A. del Pozo, and S. Rios-Insua, "A graphical decision-theoretic model for neonatal jaundice," *Medical Decision Making*, vol. 27, no. 3, pp. 250–265, 2007.

[129] G. Gybenko, "Approximation by superposition of sigmoidal functions," *Mathematics of Control, Signals and Systems*, vol. 2, no. 4, pp. 303–314, 1989.

[130] P. Artzner, F. Delbaen, J.-M. Eber, and D. Heath, "Coherent measures of risk," *Mathematical finance*, vol. 9, no. 3, pp. 203–228, 1999.

[131] M. A. Virto, J. Martín, D. R. Insua, and A. Moreno-Díaz, "Approximate Solutions of Complex Influence Diagrams through MCMC Methods," in *Probabilistic Graphical Models*.

[132] C. Abad and G. Iyengar, "Portfolio selection with multiple spectral risk constraints," *SIAM Journal on Financial Mathematics*, vol. 6, no. 1, pp. 467–486, 2015.

[133] E. Tardos and T. Wexler, "Network Formation Games and the Potential Function Method," *Algorithmic Game Theory*, pp. 487–516, 2007, ISSN: 00010782. DOI: 10.1145/1785414.1785439. arXiv: 0907.4385.

[134] H. B. McMahan, G. Holt, D. Sculley, M. Young, D. Ebner, J. Grady, L. Nie, T. Phillips, E. Davydov, D. Golovin, *et al.*, "Ad click prediction: a view from the trenches," in *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM, 2013, pp. 1222–1230.

[135] H. Cooper, C.-Y. Lin, D. Yeh, and M. Gavaudan, *System And Method For Providing An Artificially-Intelligent Graph Database*, 2018.

[136] M. Gavaudan, P. Jin, Y. Yang, C.-Y. Lin, and H. Cooper, *System and Method for Providing a Graph Protocol for Forming a Decentralized and Distributed Graph Database*, 2018.

[137] S. S. Conn, "OLTP and OLAP data integration: A review of feasible implementation methods and architectures for real time data analysis," in *Conference Proceedings - IEEE SOUTHEASTCON*, 2005.

[138] J. J. Miller, "Graph database applications and concepts with Neo4j," in *Proceedings of the Southern Association for Information Systems Conference, Atlanta, GA, USA*, 2013.

[139] G. Malewicz, M. H. Austern, A. J. C. C. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel," *Proceedings of the 28th ACM symposium on Principles of distributed computing - PODC '09*, p. 6, 2009. DOI: 10.1145/1582716.1582723. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1582716.1582723.

[140] R. R. McCune, T. Weninger, and G. Madey, "Thinking Like a Vertex," *ACM Computing Surveys*, vol. 48, no. 2, pp. 1–39, 2015, ISSN: 03600300. DOI: 10.1145/2818185. arXiv: 1507.04405.

[141] F. Funke, A. Kemper, and T. Neumann, "Benchmarking hybrid OLTP & OLAP database systems," *In BTW*, 2011.

[142] A. Kemper and T. Neumann, "HyPer: A hybrid OLTP&OLAP main memory database system based on virtual memory snapshots," in *Proceedings - International Conference on Data Engineering*, 2011, ISBN: 9781424489589. DOI: 10.1109/ICDE.2011.5767867.

[143] Y. Cao, C. Chen, F. Guo, D. Jiang, Y. Lin, B. C. Ooi, H. T. Vo, S. Wu, and Q. Xu, "ES2: A cloud data storage system for supporting both OLTP and OLAP," in *Proceedings - International Conference on Data Engineering*, 2011, ISBN: 9781424489589. DOI: 10.1109/ICDE.2011.5767881.

[144] H. Lang, T. Mühlbauer, F. Funke, P. Boncz, T. Neumann, and A. Kemper, "Data Blocks: Hybrid OLTP and OLAP on compressed storage using both vectorization and compilation," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2016, ISBN: 9781450335317. DOI: 10.1145/2882903.2882925.

[145] H. Plattner, "A common database approach for OLTP and OLAP using an in-memory column database," in *SIGMOD-PODS'09 - Proceedings of the International Conference on Management of Data and 28th Symposium on Principles of Database Systems*, 2009, ISBN: 9781605585543.

[146] M. a. Rodriguez, "The Gremlin Graph Traversal Machine and Language," *Proc. 15th Symposium on Database Programming Languages*, pp. 1–10, 2015. DOI: 10.1145/2815072.2815073. arXiv: 1508.03843.

[147] I. Rogers, "The Google Pagerank Algorithm and How It Works," 2005, [Online]. Available: http://www.sirgroane.net/google-page-rank/.

[148] L. G. Valiant, "A Bridging Model for Parallel Computation," *Commun. ACM*, vol. 33, no. 8, pp. 103–111, Aug. 1990, ISSN: 0001-0782. DOI: 10.1145/79173.79181. [Online]. Available: http://doi.acm.org/10.1145/79173.79181.

[149] V. Kalavri, S. Ewen, K. Tzoumas, V. Vlassov, V. Markl, and S. Haridi, "Asymmetry in Large-Scale Graph Analysis, Explained," in *Workshop on Graph Data Management Experiences and Systems*, 2014. [Online]. Available: https://pdfs.semanticscholar.org/2b1e/9294a2091a628c1b6b9fd6bbe148058a5cf4.pdf.

[150] Y. Tian, A. Balmin, S. A. Corsten, S. Tatikonda, and J. McPherson, "From "think like a vertex" to "think like a graph"," *Proceedings of the VLDB Endowment*, vol. 7, no. 3, pp. 193–204, 2015, ISSN: 21508097. DOI: 10.14778/2732232.2732238.

[151] Y. Simmhan, A. Kumbhare, C. Wickramaarachchi, S. Nagarkar, S. Ravi, C. Raghavendra, and V. Prasanna, "GoFFish: A sub-graph centric framework for large-scale graph analytics," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 8632 LNCS, 2014, pp. 451–462, ISBN: 9783319098722. arXiv: 1311.5949.

[152] D. Yan, J. Cheng, Y. Lu, and W. Ng, "Blogel: A Block-centric Framework for Distributed Computation on Real-world Graphs," in *Proceedings of the VLDB Endowment*, vol. 7, 2015, pp. 1981–1992, ISBN: 2150-8097. DOI: 10.14778/2733085.2733103. arXiv: arXiv:1503.07241v1.

[153] A. A. A. Donovan and B. W. Kernighan, *The Go programming language*. Addison-Wesley Professional, 2015.

[154] J. Dostert, *Thread-level resource usage measurement*, 2009.

[155] A. A. Buluç, H. Meyerhenke, I. Safro, P. Sanders, and C. Schulz, "Recent Advances in Graph Partitioning," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9220 LNCS, 2016, pp. 1–37. arXiv: arXiv:1311.3144v3.

[156] R. Cyganiak, D. Wood, and M. Lanthaler, *RDF 1.1 Concepts and Abstract Syntax*, 2014.

[157] J. Dean and S. Ghemawat, *MapReduce*, 2010. DOI: 10.1145/1629175.1629198. arXiv: 10.1.1.163.5292.

[158] R. S. Xin, J. E. Gonzalez, M. J. Franklin, I. Stoica, and E. AMPLab, "GraphX: A Resilient Distributed Graph System on Spark," *First International Workshop on Graph Data Management Experiences and Systems*, p. 2, 2013, ISSN: 0002-9513. DOI: 10.1145/2484425.2484427. arXiv: 1402.2394.

[159] C. Jung, S. Rus, B. Railing, N. Clark, and S. Pande, "Brainy: effective selection of data structures," in *PLDI '11:Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2011, ISBN: 9781450306638. DOI: 10.1145/1993498.1993509.

[160] M. De Wael, S. Marr, J. De Koster, J. B. Sartor, and W. De Meuter, "Just-in-time data structures," in *Onward!*, 2015, pp. 61–75, ISBN: 9781450336888. DOI: 10.1145/2814228.2814231.

[161] D. Costa and A. Andrzejak, "CollectionSwitch: a framework for efficient and dynamic collection selection," in *CGO*, 2018, pp. 16–26, ISBN: 9781450356176. DOI: 10.1145/3179541.3168825.

[162] B. Schiller, C. Deusser, J. Castrillon, and T. Strufe, "Compile- and run-time approaches for the selection of efficient data structures for dynamic graph analysis," *Applied Network Science*, pp. 1–22, 2016, ISSN: 2364-8228. DOI: 10.1007/s41109-016-0011-2. [Online]. Available: http://dx.doi.org/10.1007/s41109-016-0011-2.

[163] A. Kusum, I. Neamtiu, and R. Gupta, "Safe and flexible adaptation via alternate data structure representations," in *Proceedings of the 25th International Conference on Compiler Construction - CC 2016*, 2016, pp. 34–44, ISBN: 9781450342414. DOI: 10.1145/2892208.2892220.

[164] A. Dave, A. Jindal, L. Li, R. Xin, J. Gonzalez, and M. Zaharia, "GraphFrames: An Integrated API for Mixing Graph and Relational Queries," in *Proceedings of the Fourth International Workshop on Graph Data Management Experiences and Systems SE - GRADES '16*, 2016, ISBN: 9781450321389. DOI: doi:10.1145/2960414.2960416. [Online]. Available: https://event.cwi.nl/grades/2016/02-Dave.pdf%20http://dx.doi.org/10.1145/2960414.2960416.

[165] J. Webber, "A Programmatic Introduction to Neo4J," *Proceedings of the 3rd Annual Conference on Systems, Programming, and Applications: Software for Humanity*, pp. 217–218, 2012. DOI: 10.1145/2384716.2384777. [Online]. Available: http://doi.acm.org/10.1145/2384716.2384777.

[166] O. Green and D. A. Bader, "cuSTINGER: Supporting dynamic graph algorithms for GPUs," *2016 IEEE High Performance Extreme Computing Conference, HPEC 2016*, 2016. DOI: 10.1109/HPEC.2016.7761622.

[167] S. Che, B. M. Beckmann, and S. K. Reinhardt, "BelRed: Constructing GPGPU graph applications with software building blocks," *2014 IEEE High Performance Extreme Computing Conference, HPEC 2014*, 2014. DOI: 10.1109/HPEC.2014.7040961. [Online]. Available: https://pdfs.semanticscholar.org/1e22/d0867db4af2c1887bbfcd40a7b527e9eb8e9.pdf.

[168] A. J. Booker, J. E. Dennis, P. D. Frank, D. B. Serafini, V. Torczon, and M. W. Trosset, "A rigorous framework for optimization of expensive functions by surrogates," *Structural Optimization*, 1999, ISSN: 09344373. DOI: 10.1007/BF01197708.

[169] K. Langed, A. R. Hunter, and I. Yang, "Optimization Transfer Using Surrogate Objective Functions," *Journal of Computational and Graphical Statistics*, 2000, ISSN: 15372715. DOI: 10.1080/10618600.2000.10474858.

[170] P. Barceló, L. Libkin, A. W. Lin, and P. T. Wood, "Expressive Languages for Path Queries over Graph-Structured Data," *ACM Trans. Database Syst.*, vol. 37, no. 4, 31:1–31:46, 2012, ISSN: 0362-5915. DOI: 10.1145/2389241.2389250. [Online]. Available: http://doi.acm.org/10.1145/2389241.2389250.

[171] R. Angles, M. Arenas, P. Barceló, A. Hogan, J. Reutter, and D. Vrgoč, "Foundations of modern query languages for graph databases," *ACM Computing Surveys (CSUR)*, vol. 50, no. 5, p. 68, 2017.

[172] L. Libkin, W. Martens, and D. Vrgoč, "Querying Graphs with Data," *Journal of the ACM*, vol. 63, no. 2, pp. 1–53, 2016, ISSN: 00045411. DOI: 10.1145/2850413.

[173] F. Holzschuher and R. Peinl, "Performance of graph query languages: comparison of Cypher, Gremlin and native access in Neo4j," *16th International Conference on Extending Database Technology, EDBT' 13*, no. March 2013, pp. 195–204, 2013. DOI: 10.1145/2457317.2457351. [Online]. Available: http://dl.acm.org/citation.cfm?id=2457351.

[174] B. Gallagher, "Matching Structure and Semantics : A Survey on Graph-Based Pattern Matching," *AIII FS*, vol. 6, pp. 45–53, 2006.

[175] B. Schiller, J. Castrillon, and T. Strufe, "Efficient data structures for dynamic graph analysis," in *11th International Conference on Signal-Image Technology & Internet-Based Systems*, 2015.

[176] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[177] P. Erdos, A. Renyi, P. Erd\Hos, and A. Rényi, "On the evolution of random graphs," *Publications of the Mathematical Institute of the Hungarian Academy of Sciences*, vol. 5, no. 1, pp. 17–60, 1960.

[178] E. C. Foster and S. Godbole, *Database systems: a pragmatic approach*. Apress, 2016.

[179] G. Tillmann, "Introduction to Usage-Driven Database Design," in *Usage-Driven Database Design: From Logical Data Modeling through Physical Schema Definition*. Berkeley, CA: Apress, 2017, pp. 3–12, ISBN: 978-1-4842-2722-0. DOI: 10.1007/978-1-4842-2722-0_1.

[180] O. van Rest, S. Hong, J. Kim, X. Meng, H. Chafi, O. van Rest, S. Hong, J. Kim, X. Meng, and H. Chafi, "PGQL: Property Graph Query Language," *Grades*, no. Grades, pp. 0–5, 2016. DOI: 10.1145/2960414.2960421. [Online]. Available: https://docs.oracle.com/cd/E56133%7B%5C_%7D01/1.2.0/tutorials/graph-pattern-matching.html.

[181] S. Zhang, J. Yang, and W. Jin, "SAPPER: Subgraph indexing and approximate matching in large graphs," *Proceedings of the VLDB Endowment*, 2010, ISSN: 21508097. DOI: 10.14778/1920841.1920988.

[182] K. Riesen and H. Bunke, "Approximate graph edit distance computation by means of bipartite graph matching," *Image and Vision Computing*, 2009, ISSN: 02628856. DOI: 10.1016/j.imavis.2008.04.004.

[183] Y. Tian and J. M. Patel, "TALE: A tool for approximate large graph matching," in *Proceedings - International Conference on Data Engineering*, 2008, ISBN: 9781424418374. DOI: 10.1109/ICDE.2008.4497505.

[184] M. Gori, M. Maggini, and L. Sarti, "Exact and approximate graph matching using random walks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2005, ISSN: 01628828. DOI: 10.1109/TPAMI.2005.138.

[185] A. Bonifati, R. Ciucanu, and A. Lemay, "Learning path queries on graph databases," in *18th International Conference on Extending Database Technology (EDBT)*, 2015.

[186] P. Barceló and M. Romero, "The complexity of reverse engineering problems for conjunctive queries," *arXiv preprint arXiv:1606.01206*, 2016.

[187] M. Arenas, G. I. Diaz, and E. V. Kostylev, "Reverse Engineering SPARQL Queries," in *Proceedings of the 25th International Conference on World Wide Web - WWW '16*, 2016, ISBN: 9781450341431. DOI: 10.1145/2872427.2882989.

[188] J. Bergstra and D. Warde-farley, "Theano : Deep Learning on GPUs with Python," *Journal of Machine Learning Research*, 2011, ISSN: 15731561. DOI: 10.1007/s10886-016-0765-0.

[189] Chollet François, *Keras: The Python Deep Learning library*, 2015. DOI: 10.1086/316861.

[190] F. L. Bauer, "Computational Graphs and Rounding Error," *SIAM Journal on Numerical Analysis*, vol. 11, no. 1, pp. 87–96, 2005, ISSN: 0036-1429. DOI: 10.1137/0711010.

[191] T. Kalisky, S. Sreenivasan, L. A. Braunstein, S. V. Buldyrev, S. Havlin, and H. E. Stanley, "Scale-free networks emerging from weighted random graphs," *Physical Review E - Statistical, Nonlinear, and Soft Matter Physics*, 2006, ISSN: 15393755. DOI: 10.1103/PhysRevE.73.025103.

[192] A. Mnih and K. Gregor, "Neural Variational Inference and Learning in Belief Networks," in *International Conference on Machine Learning*, 2014, pp. 1791–1799. arXiv: 1402.0030. [Online]. Available: http://arxiv.org/abs/1402.0030.

[193] A. Mnih and D. J. Rezende, "Variational inference for Monte Carlo objectives," *arXiv preprint arXiv:1602.06725*, 2016.

[194] H. Tong, C. Faloutsos, and J. Y. Pan, "Fast random walk with restart and its applications," in *Proceedings - IEEE International Conference on Data Mining, ICDM*, 2006, ISBN: 0769527019. DOI: 10.1109/ICDM.2006.70.

[195] C. C. Aggarwal, *Recommender Systems*. Springer International Publishing, 2016.

[196] X. Chew and V. C. Gorgonia, *Gorgonia*, 2019. [Online]. Available: https://github.com/gorgonia/gorgonia.

[197] A. Wynn, *Xbox One API*, 2018. [Online]. Available: https://xboxapi.com/ (visited on 01/05/2018).

[198] B. Selman, H. A. Kautz, and B. Cohen, "Noise strategies for improving local search," in *AAAI*, vol. 94, 1994, pp. 337–343.

[199] D. McAllester, B. Selman, and H. Kautz, "Evidence for invariants in local search," in *AAAI/IAAI*, Rhode Island, USA, 1997, pp. 321–326.

[200] Y. Song, S. Dixon, and M. Pearce, "A survey of music recommendation systems and future perspectives," *9th International Symposium on Computer Music Modeling and Retrieval*, 2012.

[201] C. A. Gomez-Uribe and N. Hunt, "The Netflix Recommender System," *ACM Transactions on Management Information Systems*, vol. 6, no. 4, pp. 1–19, 2015, ISSN: 2158656X. DOI: 10.1145/2843948. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2869770.2843948.

[202] Microsoft, *Xbox Live | Xbox*, 2002. [Online]. Available: https://www.xbox.com/en-US/live (visited on 01/05/2018).

[203] B. Kuchera, *The anatomy of a review bombing campaign - Polygon*, 2017. [Online]. Available: https://www.polygon.com/2017/10/4/16418832/pubg-firewatch-steam-review-bomb (visited on 05/05/2018).

[204] N. Grayson, *Total War Game Gets Review Bombed On Steam Over Women Generals*, 2018. [Online]. Available: https://steamed.kotaku.com/total-war-game-gets-review-bombed-on-steam-over-women-g-1829283785 (visited on 10/02/2018).

[205] J. Leino and K.-J. Räihä, "Case Amazon: Ratings and Reviews as Part of Recommendations," *Proceedings of the 2007 ACM conference on Recommender systems - RecSys '07*, p. 137, 2007. DOI: 10.1145/1297231.1297255. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1297231.1297255.

[206] J. Tang, X. Hu, and H. Liu, "Social recommendation: a review," *Social Network Analysis and Mining*, 2013, ISSN: 18695469. DOI: 10.1007/s13278-013-0141-9.

[207] H. Ma, H. Yang, M. R. Lyu, and I. King, "SoRec : Social Recommendation Using Probabilistic Matrix Factorization," *Proceeding of the 17th ACM conference on Information and knowledge management*, 2008. DOI: 10.1145/1458082.1458205.

[208] I. Konstas, V. Stathopoulos, and J. M. Jose, "On Social Networks and Collaborative Recommendation," in *Proceedings of the 32Nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2009, ISBN: 978-1-60558-483-6. DOI: 10.1145/1571941.1571977.

[209] I. King, M. R. Lyu, and H. Ma, "Introduction to social recommendation," in *Proceedings of the 19th international conference on World wide web - WWW '10*, 2010, ISBN: 9781605587998. DOI: 10.1145/1772690.1772927.

[210] Y. Fujiwara, M. Nakatsuji, T. Yamamuro, H. Shiokawa, and M. Onizuka, "Efficient personalized pagerank with accuracy assurance," in *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2012, pp. 15–23, ISBN: 9781450314626. DOI: 10.1145/2339530.2339538.

[211] T. H. Haveliwala, "Topic-sensitive PageRank," in *Proceedings of the eleventh international conference on World Wide Web*, ACM, 2002, pp. 517–526, ISBN: 1581134495. DOI: 10.1145/511446.511513. [Online]. Available: http://portal.acm.org/citation.cfm?doid=511446.511513.

[212] M. Al Hasan and M. J. Zaki, "A survey of link prediction in social networks," in *Social network data analytics*, Springer, 2011, pp. 243–275.

[213] D. Liben-Nowell and J. Kleinberg, "The link-prediction problem for social networks," *Journal of the American Society for Information Science and Technology*, vol. 58, no. 7, pp. 1019–1031, 2007, ISSN: 15322882. DOI: 10.1002/asi.20591. arXiv: 0803.1716.

[214] ——, "The Link Prediction Problem for Social Networks," *Proceedings of the Twelfth Annual ACM International Conference on Information and Knowledge Management (CIKM)*, 2003, ISSN: 1532-2882. DOI: 10.1002/asi.v58:7. arXiv: arXiv:1010.0725v1.

[215] L. Lu and T. Zhou, "Link Prediction in Complex Networks: A Survey," *Physica A: Statistical Mechanics and its Applications*, 2017, ISSN: 03784371. DOI: 10.1016/j.physa.2010.11.027. arXiv: 1010.0725.

[216] M. Jamali and M. Ester, "TrustWalker: a random walk model for combining trust-based and item-based recommendation," *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 397–406, 2009. DOI: citeulike-article-id:5151320.

[217] M. Gori and A. Pucci, "ItemRank: A Random-Walk Based Scoring Algorithm for Recommender Engines," *IJCAI*, vol. 7, pp. 2766–2771, 2007.

[218] R. Levin, H. Abassi, and U. Cohen, "Guided Walk: A Scalable Recommendation Algorithm for Complex Heterogeneous Social Networks," *RecSys*, 2016, ISSN: 16130073. DOI: 10.1017/CBO9781107415324.004. arXiv: arXiv:1011.1669v3.

[219] L. Backstrom and J. Leskovec, "Supervised random walks," in *Proceedings of the fourth ACM international conference on Web search and data mining - WSDM '11*, 2011, ISBN: 9781450304931. DOI: 10.1145/1935826.1935914. arXiv: 1011.4071.

[220] J. Hamari, "Framework for Designing and Evaluating Game Achievements," *Proceedings of DiGRA 2011 Conference: Think Design Play*, p. 20, 2011, ISSN: ISSN 2342-9666. DOI: 10.1.1.224.9966. arXiv: 11307.59151. [Online]. Available: http://www.mendeley.com/catalog/framework-designing-evaluating-game-achievements/.