SDM-Networks 2016
The Third SDM Workshop on Mining Networks and Graphs: A Big Data Analytic Challenge

May 7, 2016, Miami, Florida, USA

In conjunction with

2016 SIAM International Conference on Data Mining (SDM16)

Workshop Chairs

Maleq Khan (Virginia Tech)
Christine Klymko (Lawrence Livermore National Laboratory)
Larry Holder (Washington State University)

Preface

Real-world applications give rise to networks that are unstructured and often comprised of several components. Furthermore, they can support multiple dynamical processes that shape the network over time. Network science refers to the broad discipline that seeks to understand the underlying principles that govern the synthesis, analysis and co-evolution of networks. In some cases, the data relevant for mining patterns and making decisions comes from multiple heterogeneous sources and streams in over time. Graphs are a popular representation for such data because of their ability to represent different entity and relationship types, including the temporal relationships necessary to represent the dynamics of a data stream. However, fusing such heterogeneous data into a single graph or multiple related graphs and mining them are challenging tasks. Emerging massive data has made such tasks even more challenging.

This workshop will bring together researchers and practitioners in the field to deal with the emerging challenges in processing and mining large-scale networks. Such networks can be directed as well as undirected, they can be labeled or unlabeled, weighted or unweighted, and static or dynamic. Networks of networks are also of interest. Specific scientific topics of interest for this meeting include mining for patterns of interest in networks, efficient algorithms (sequential/parallel, exact/approximation) for analyzing network properties, methods and systems for processing large networks (i.e., Map-Reduce, GraphX, Giraph, etc.), use of linear algebra and numerical analysis for mining complex networks, database techniques for processing networks, and fusion of heterogeneous data sources into graphs. Another particular topic of interest is to couple structural properties of networks to the dynamics over networks, e.g., contagions.

This day long workshop will feature a keynote address, several invited talks, contributed papers, and a panel discussion on open problems and directions for future research.

Program Committee

- Leman Akoglu (SUNY Stony Brook)
- Albert Bifet (University of Waikato and Noah's Ark Lab)
- Rajmonda Caceres (MIT Lincoln Lab)
- Sutanay Choudhury (Pacific Northwest National Lab)
- Bill Eberle (Tennessee Technological University)
- Mohammad Al Hasan (Indiana University - Purdue University Indianapolis)
- David Kempe (University of Southern California)
- Christopher Kuhlman (Virginia Tech)
- Kamesh Madduri (Pennsylvania State University)
- Madhav Marathe (Virginia Tech)
- Ali Pinar (Sandia National Laboratories)
- Anil Vullikanti (Virginia Tech)
- Yinghui Wu (Washington State University)

Keynote Speaker

TBD

Invited Speakers

TBD

List of Papers

- *On the Geometry and Extremal Properties of the Edge-Degeneracy Model*
  Nicolas Kim, Dane Wilburne, Sonja Petrovic, and Alessandro Rinaldo
- *Collaborative SVM Classification in Skewed Peer-to-Peer Networks*
  Umer Khan, Alexandros Nanopoulos, and Lars Schmidt-Thieme
- *Towards Scalable Graph Analytics on Time Dependent Graphs*
  Suraj Poudel, Roger Pearce, and Maya Gokhale
- *GSK: Graph Sparsification as a Knapsack problem formulation*
  Hongyuan Zhan and Kamesh Madduri
- *Trust from the past: Bayesian Personalized Ranking based Link Prediction in Knowledge Graphs*
  Baichuan Zhang, Sutanay Choudhury, Mohammad Al Hasan, Xia Ning, Khushbu Agarwal, Sumit Purohit, and Paola Pesantez Cabrera

On the Geometry and Extremal Properties of the Edge-Degeneracy Model*

Nicolas Kim†‡ Dane Wilburne†§ Sonja Petrović§ Alessandro Rinaldo‡

Abstract

The edge-degeneracy model is an exponential random graph model that uses the graph degeneracy, a measure of the graph's connection density, and number of edges in a graph as its sufficient statistics. We show this model is relatively well-behaved by studying the statistical degeneracy of this model through the geometry of the associated polytope.

Keywords exponential random graph model, degeneracy, k-core, polytope

1 Introduction

Statistical network analysis is concerned with developing statistical tools for assessing, validating and modeling the properties of random graphs, or networks. The very first step of any statistical analysis is the formalization of a statistical model, a collection of probability distributions over the space of graphs (usually, on a fixed number of nodes n), which will serve as a reference model for any inferential tasks one may want to perform. Statistical models are in turn designed to be interpretable and, at the same time, to be capable of reproducing the network characteristics pertaining to the particular problem at hand. Exponential random graph models, or ERGMs, are arguably the most important class of models for networks with a long history. They are especially useful when one wants to construct models that resemble the observed network, but without the need to define an explicit network formation mechanism. In the interest of space, we single out classical references [2], [4], [9] and a recent review paper [8].

Central to the specification of an ERGM is the choice of sufficient statistic, a function on the space of graphs, usually vector-valued, that captures the particular properties of a network that are of scientific interest. Common examples of sufficient statistics are the number of edges, triangles, or k-stars, the degree sequence, etc; for an overview, see [8]. The choice of a sufficient statistic is not to be taken for granted: it depends on the application at hand and at the same time it dictates the statistical and mathematical behavior of the ERGM. While there is not a general classification of 'good' and 'bad' network statistics, some lead to models that behave better asymptotically than others, so that computation and inference on large networks can be handled in a reliable way.

In an ERGM, the probability of observing any given graph depends on the graph only through the value of its sufficient statistic, and is therefore modulated by how much or how little the graph expresses those properties captured by the sufficient statistics. As there is virtually no restriction on the choice of the sufficient statistics, the class of ERGMs therefore possesses remarkable flexibility and expressive power, and offers, at least in principle, a broadly applicable and statistically sound means of validating any scientific theory on real-life networks. However, despite their simplicity, ERGMs are also difficult to analyze and are often thought to behave in pathological ways, e.g., give significant mass to extreme graph configurations. Such properties are often referred to as degeneracy; here we will refer to it as *statistical degeneracy* [10] (not to be confused with graph degeneracy below). Further, their asymptotic properties are largely unknown, though there has been some recent work in this direction; for example, [6] offer a variation approach, while in some cases it has been shown that their geometric properties can be exploited to reveal their extremal asymptotic behaviors [22], see also [18]. These types of results are interesting not only mathematically, but have statistical value: they provide a catalogue of extremal behaviors as a function of the model parameters and illustrate the extent to which statistical degeneracy may play a role in inference.

In this article we define and study the properties of the ERGM whose sufficient statistics vector consists of two quantities: the edge count, familiar to and often used in the ERGM family, and the graph degeneracy, novel to the statistics literature. (These quantities may be scaled appropriately, for purpose of asymptotic considerations; see Section 2.) As we will see, graph degeneracy arises from the graph's core structure, a property that is new to the ERGM framework [11], but is a natural connectivity statistic that gives a sense of how densely connected the most important actors in the network are. The core structure of a graph

*Partially supported by AFOSR grant #FA9550-14-1-0141.

†Equal contribution

‡Department of Statistics, Carnegie Mellon University, Email: nicolask@stat.cmu.edu, arinaldo@cmu.edu.

§Department of Applied Mathematics, Illinois Institute of Technology, Email: dwilburn@hawk.iit.edu, sonja.petrovic@iit.edu.

(see Definition 2.1) is of interest to social scientists and other researchers in a variety of applications, including the identification and ranking of influencers (or "spreaders") in networks (see [14] and [1]), examining robustness to node failure, and for visualization techniques for large-scale networks [5]. The degeneracy of a graph is simply the statistic that records the largest core.

Cores are used as descriptive statistics in several network applications (see, e.g., [16]), but until recently, very little was known about statistical inference from this type of graph property: [11] shows that cores are unrelated to node degrees and that restricting graph degeneracy yields reasonable core-based ERGMs. Yet, there are currently no rigorous statistical models for networks in terms of their degeneracy. The results in this paper thus add a dimension to our understating of cores by exhibiting the behavior of the joint edge-degeneracy statistic within the context of the ERGM that captures it, and provide extremal results critical to estimation and inference for the edge-degeneracy model.

We define the edge-degeneracy ERGM in Section 2, investigate its geometric structure in Sections 3, 4, and 5, and summarize the relevance to statistical inference in Section 6.

2 The edge-degeneracy (ED) model

This section presents the necessary graph-theoretical tools, establishes notation, and introduces the ED model. Let $\mathcal{G}_n$ denote the space of (labeled, undirected) simple graphs on n nodes, so $|\mathcal{G}_n| = 2^{\binom{n}{2}}$.

To define the family of probability distributions over $\mathcal{G}_n$ comprising the ED model, we first define the degeneracy statistic.

DEFINITION 2.1. *Let $G = (V, E)$ be a simple, undirected graph. The* k-core *of G is the maximal subgraph of G with minimum degree at least k. Equivalently, the k-core of G is the subgraph obtained by iteratively deleting vertices of degree less than k. The* graph degeneracy *of G, denoted* $\mathrm{degen}(G)$*, is the maximum value of k for which the k-core of G is non-empty.*

This idea is illustrated in Figure 1, which shows a graph G and its 2-core. In this case, $\mathrm{degen}(G) = 4$.

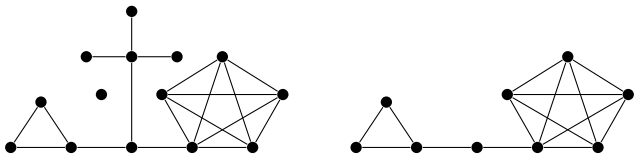


Figure 1: A small graph G (left) and its 2-core (right). The degeneracy of this graph is 4.

The *edge-degeneracy* ERGM is the statistical model on $\mathcal{G}_n$ whose sufficient statistics are the rescaled graph degeneracy and the edge count of the observed graph. Concretely, for $G \in \mathcal{G}_n$ let

$$t(G) = \left(E(G) / \binom{n}{2}, \mathrm{degen}(G)/(n-1) \right), \tag{2.1}$$

where $E(G)$ is the number of edges of G. The ED model on $\mathcal{G}_n$ is the ERGM $\{P_{n,\theta}, \theta \in \mathbb{R}^2\}$, where

$$P_{n,\theta}(G) = \exp\{\langle \theta, t(G) \rangle - \psi(\theta)\} \tag{2.2}$$

is the probability of observing the graph $G \in \mathcal{G}_n$ for the choice of model parameter $\theta \in \mathbb{R}^2$. The log-partition function $\psi\colon \mathbb{R}^2 \to \mathbb{R}$, given by $\psi(\theta) = \sum_{G \in \mathcal{G}_n} e^{\langle \theta, t(G) \rangle}$ serves as a normalizing constant, so that probabilities add up to 1 for each choice of θ (notice that $\psi(\theta) < \infty$ for all θ, as $\mathcal{G}_n$ is finite).

Notice that different choices of $\theta = (\theta_1, \theta_2)$ will lead to rather different distributions. For example, for large and positive values of θ_1 and θ_2 the probability mass concentrates on dense graphs, while negative values of the parameters will favor sparse graphs. More interestingly, when one parameter is positive and the other is negative, the model will favor configurations in which the edge and degeneracy count will be balanced against each other. Our results in Section 5 will provide a catalogue of such behaviors in extremal cases and for large n.

The normalization of the degeneracy and the edge count in (2.1) and the presence of the coefficient $\binom{n}{2}$ in the ED probabilities (2.2) are to ensure a non-trivial limiting behavior as $n \to \infty$, since $E(G)$ and $\mathrm{degen}(G)$ scale differently in n (see, e.g., [6] and [22]). This normalization is not strictly necessary for our theoretical results to hold. However, the ED model, like most ERGMs, is not consistent, thus making asymptotic considerations somewhat problematic.

LEMMA 2.1. *The edge-degeneracy model is an ERGM that is not consistent under sampling, as in [21].*

Proof. The range of graph degeneracy values when going from a graph with n vertices to one with $n+1$ vertices depends on the original graph; e.g. if there is a 2-star that is not a triangle in a graph with three vertices, the addition of another vertex can form a triangle and increase the graph degeneracy to 2, but if there was not a 2-star then there is no way to increase the graph degeneracy. Since the range is not constant, this ERGM is not consistent under sampling.

Thus, as the number of vertices n grows, it is important to note the following property of the ED model, not

uncommon in ERGMs: inference on the whole network cannot be done by applying the model to subnetworks.

In the next few sections we will study the geometry of the ED model as a means to derive some of its asymptotic properties. The use of polyhedral geometry in the statistical analysis of discrete exponential families is well established: see, e.g., [2], [4], [7], [20], [19].

3 Geometry of the ED model polytope

The edge-degeneracy ERGM (2.2) is a discrete exponential family, for which the geometric structure of the model carries important information about parameter estimation including existence of maximum likelihood estimate (MLE) - see above mentioned references. This geometric structure is captured by the *model polytope*.

The model polytope $\mathcal{P}_n$ of the ED model on $\mathcal{G}_n$ is the convex hull of the set of all possible edge-degeneracy pairs for graphs in $\mathcal{P}_n$. In symbols,

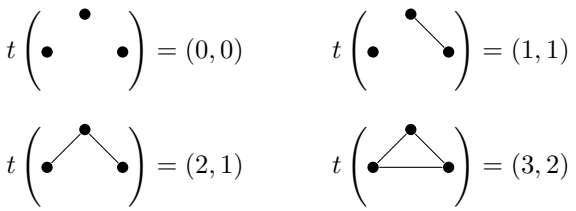
$$\mathcal{P}_n := \text{conv}\left\{(E(G), \text{degen}(G)), G \in \mathcal{G}_n\right\} \subset \mathbb{R}^2.$$

Note the use of the unscaled version of the sufficient statistics in defining the model polytope. In this section, the scaling used in model definition (2.1) has little impact on shape of $\mathcal{P}_n$, thus - for simplicity of notation - we do not include it in the definition of $\mathcal{P}_n$. The scaling factors will be re-introduced, however, when we consider the normal fan and the asymptotics in Section 4.

In the following, we characterize the geometric properties of $\mathcal{P}_n$ that are crucial to statistical inference. First, we arrive at a startling result, Proposition 3.1, that every integer point in the model polytope is a realizable statistic. One implication of this is obtained in conjunction with other asymptotic results discussed below. Second, Proposition 3.3 implies that the observed network statistics will almost surely lie in the relative interior of the model polytope, which is an important property because estimation algorithms are guaranteed to behave well when off the boundary of the polytope. This also implies that the MLE for the edge-degeneracy ERGM exists almost surely for large graphs. In other words, there are very few network observations that can lead to statistical degeneracy, that is, bad behavior of the model for which some ERGMs are famous. That behavior implies that a subset of the natural parameters is non-estimable, making complete inference impossible. Thus it being avoided by the edge-degeneracy ERGM is a desirable outcome. In summary, Propositions 3.3, 3.1, 3.4 and Theorem 3.1 completely characterize the geometry of $\mathcal{P}_n$ and thus solve [17, Problem 4.3] for this particular ERGM. Remarkably, this problem—although critical for our understanding of reliability of inference for such models– has not been solved for most ERGMs except, for example, the beta model [20], which relied heavily on known graph-theoretic results.

Let us consider $\mathcal{P}_n$ for some small values of n. The polytope $\mathcal{P}_{10}$ is plotted in Figure 2.

The case $n = 3$. There are four non-isomorphic graphs on 3 vertices, and each gives rise to a distinct edge-degeneracy vector:



Hence $\mathcal{P}_3 = \text{conv}\{(0,0),(1,1),(2,1),(3,2)\}$. Note that in this case, each realizable edge-degeneracy vector lies on the boundary of the model polytope. We will see below that $n = 3$ is the unique value of n for which there are no realizable edge-degeneracy vectors contained in the relative interior of $\mathcal{P}_n$.

The case $n = 4$. On 4 vertices there are 11 non-isomorphic graphs but only 8 distinct edge-degeneracy vectors. Without listing the graphs, the edge-degeneracy vectors are:

$$(0,0),(1,1),(2,1),(3,1),(3,2),(4,2),(5,2),(6,3).$$

Here we pause to make the simple observation that $\mathcal{P}_n \subset \mathcal{P}_{n+1}$ always holds. Indeed, every realizable edge-degeneracy vector for graphs on n vertices is also realizable for graphs on $n+1$ vertices, since adding a single isolated vertex to a graph affects neither the number of edges nor the graph degeneracy.

The case $n = 5$. There are 34 non-isomorphic graphs on $n = 5$ vertices but only 15 realizable edge-degeneracy vectors. They are:

$$(0,0),(1,1),(2,1),(3,1),(3,2),(4,2),(5,2),(6,3),$$

$$(4,1),(6,2),(7,2),(7,3),(8,3),(9,3),(10,4),$$

where the pairs listed on the top row are contained in $\mathcal{P}_4$ and the pairs on the second row are contained in $\mathcal{P}_5 \setminus \mathcal{P}_4$. Here we make the observation that the proportion of realizable edge-degeneracy vectors lying on the interior of the $\mathcal{P}_n$ seems to be increasing with n. This phenomenon is addressed in Proposition 3.3 below. Figure 2 depicts the integer points that define $\mathcal{P}_{10}$.

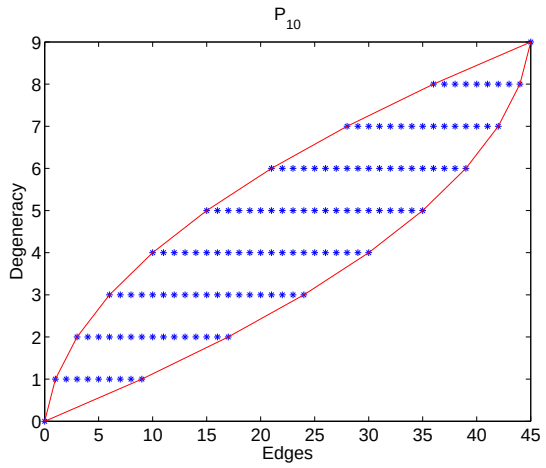


Figure 2: The integer points that define the model polytope $\mathcal{P}_{10}$.

The case for general n. It is well known in the theory of discrete exponential families that the MLE exists if and only if the average sufficient statistic of the sample lies in the relative interior of the model polytope. This leads us to investigate which pairs of integer points correspond to realizable edge-degeneracy vectors.

PROPOSITION 3.1. *Every integer point contained in $\mathcal{P}_n$ is a realizable edge-degeneracy vector.*

Proof. Suppose that G is a graph on n vertices with $\text{degen}(G) = d \leq n-1$. Let $U_n(d)$ be the minimum number of edges over all such graphs and let $L_n(d)$ be the maximum number of edges over all such graphs. Our strategy will be to show that for all e such that $U_n(d) \leq e \leq L_n(d)$, there exists a graph G on n vertices such that $\text{degen}(G) = d$ and $E(G) = e$.

First, observe that if $\text{degen}(G) = d$, then there are at least $d+1$ vertices of G in the d-core. Using this observation, it is not difficult to see that $U_n(d) = \binom{d+1}{2}$. This is the minimum number of edges required to construct a graph with a non-empty d-core; hence, the upper boundary of $\mathcal{P}_n$ consists of the points $(\binom{d+1}{2}, d)$ for $0 \leq d \leq n-1$. Further, it is clear that there is exactly one graph (up to isomorphism) corresponding to the edge-degeneracy vector $(\binom{d+1}{2}, d)$; it is the graph

$$K_{d+1} + \underbrace{K_1 + \ldots + K_1}_{n-d-1 \text{ times}}, \tag{3.3}$$

i.e., the complete graph on $d+1$ vertices along with $n-d-1$ isolated vertices.

It is an immediate consequence of [11, Proposition 11] that $L_n(d) = \binom{d+1}{2} + (n-d-1)\cdot d$. Thus, for each e such that $U_n(d) = \binom{d+1}{2} < e < \binom{d+1}{2} + (n-d-1)\cdot d = L_n(d)$, we must show how to construct a graph G on n vertices with graph degeneracy d and e edges. Call the resulting graph $G_{n,d,e}$. To construct $G_{n,d,e}$, start with the graph in 3.3. Label the isolated vertices $v_1, \ldots, v_{n-d-1}$. For each j such that $1 \leq j \leq e - \binom{d+1}{2}$, add the edge e_j by making the vertex v_i such that $i \equiv j \mod n-d-1$ adjacent to an arbitrary vertex of K_{d+1}. This process results in a graph with exactly $e = \binom{d+1}{2} + j$ edges, and since $j < (n-d-1)\cdot d$, our construction guarantees that we have not increased the graph degeneracy. Hence, we have constructed $G_{n,d,e}$. Combined with Lemma 3.1 below, this proves the desired result.

The preceding proof also shows the following:

PROPOSITION 3.2. *$\mathcal{P}_n$ contains exactly*

$$\sum_{d=0}^{n-1} [(n-d-1)\cdot d + 1] \tag{3.4}$$

integer points. This is the number of realizable edge-degeneracy vectors for every n.

The following property is useful throughout:

LEMMA 3.1. *$\mathcal{P}_n$ is rotationally symmetric.*

Proof. For $n \geq 3$ and $d \in \{1, \ldots, n-1\}$,

$$L_n(d) - L_n(d-1) = U_n(n-d) - U_n(n-d-1).$$

Note that the center of rotation is the point $((n-1)n/4, (n-1)/2)$. The rotation is 180 degrees around that point.

As mentioned above, the following nice property of the ED model polytope implies that the MLE for the ED model exists almost surely for large graphs.

PROPOSITION 3.3. *Let p_n denote the proportion of realizable edge-degeneracy vectors that lie on the relative interior of $\mathcal{P}_n$. Then,*

$$\lim_{n\to\infty} p_n = 1.$$

Proof. This result follows from analyzing the formula in 3.4 and uses the following lemma:

LEMMA 3.2. *There are $2n-2$ realizable lattice points on the boundary of $\mathcal{P}_n$, and each is a vertex of the polytope.*

Proof. We know that there are $2n-2$ lattice points on the boundary of $\mathcal{P}_n$. We show that they are all vertices of $\mathcal{P}_n$. Note that these will be the *only* vertices of $\mathcal{P}_n$, since $\mathcal{P}_n$ is the closure of the convex hull of a set of

lattice points S, so if S contains any lattice points, they will certainly all be in $\mathcal{P}_n$.

First, since $\mathcal{P}_n \subseteq \mathbb{Z}^+ \times \mathbb{Z}^+$, and $(0,0) \in \mathcal{P}_n$ for all n, we know that $(0,0)$ must be a vertex of $\mathcal{P}_n$. By the rotational symmetry of $\mathcal{P}_n$, $(n-1,(n-1)n/2)$ must be a vertex, too.

It is sufficient to show that $\Delta_U(d) := U_n(d) - U_n(d-1)$ satisfies $\Delta_U(d) \neq \Delta_U(d-1)$ for all $d \in \{2,3,\ldots,n-1\}$; this is because of the rotational symmetry of $\mathcal{P}_n$. Since $\Delta_U(d) = d$, and $d \neq d-1$ for any d, each point on the upper boundary must be a vertex of the polytope. This proves the lemma.

To prove the proposition, we then compute:

$$p_n = \frac{\sum_{d=0}^{n-1}\left[(n-d-1)\cdot d+1\right]-(2n-2)}{\sum_{d=0}^{n-1}\left[(n-d-1)\cdot d+1\right]} \to 1.$$

3.1 Extremal points of $\mathcal{P}_n$. Now we turn our attention to the problem of identifying the graphs corresponding to extremal points of $\mathcal{P}_n$. Clearly, the boundary point $(0,0)$ is uniquely attained by the empty graph $\bar{K}_n$ and the boundary point $(\binom{n}{2}, n-1)$ is uniquely attained by the compete graph K_n. The proof of Proposition 3.1 shows that the unique graph corresponding to the upper boundary point $(U_n(d), d)$ is a complete graph on $d+1$ vertices union $n-d-1$ isolated vertices. The lower boundary graphs are more complicated, but the graphs corresponding to two of them are classified in the following proposition.

PROPOSITION 3.4. *The graphs corresponding to the lower boundary point $(L_n(1),1)$ of $\mathcal{P}_n$ are exactly the trees on n vertices. The unique graph corresponding to the lower boundary point $(L_n(n-2), n-2)$ is the complete graph on n vertices minus an edge.*

Proof. First we consider graphs with edge-degeneracy vector $(L_n(1),1)$. By the formula given in the proof of Proposition 3.1, $L_n(1) = \binom{2}{2} + (n-1-1) = n-1$. A graph with degeneracy 1 must be acyclic, since otherwise it would have degeneracy at least 2. By a well known classification theorem for trees, an acyclic graph on n vertices with $n-1$ edges is a tree. Similarly, a graph corresponding to the lower boundary point $(L_n(n-2), n-2)$ has $\binom{n-1}{2} + (n-(n-2)-1)(n-2) = \binom{n}{2} - 1$ edges. There is only one such graph: the complete graph on n vertices minus an edge.

The graphs corresponding to extremal points $(L_n(d), d)$ for $2 \leq d \leq n-3$ are called *maximally d-degenerate* graphs and were studied extensively in [3]. Such graphs have many interesting properties, but are quite difficult to fully classify or enumerate.

In the following theorem, we show that the lower edge of $\mathcal{P}_n$ is like a mirrored version of the upper edge. This partially characterizes the remaining maximally d-degenerate graphs.

THEOREM 3.1. *Consider $G(U_n(d)) \in \mathcal{G}_n$ to be the unique graph on n nodes with degeneracy d that has the minimum number of edges, given by $U_n(d)$. Similarly, let $G(L_n(d)) \subset \mathcal{G}_n$ be the set of graphs on n nodes with degeneracy d that have the maximum number of edges, given by $L_n(d)$. Then, for all $d \in \{0,1,\ldots,n-1\}$,*

$$\overline{G(U_n(d))} \in G(L_n(n-(d+1))).$$

Proof. As we know, $G(U_n(d)) = K_{d+1} + \overline{K_{n-(d+1)}}$. Taking the complement,

$$\overline{G(U_n(d))} = \overline{K_{d+1}} \cup K_{n-(d+1)}. \tag{3.5}$$

We only need to show that this has $L_n((n-1)-d)$ edges and graph degeneracy $(n-1)-d$.

It is straightforward to show that (3.5) has the right number of edges. Since $G(U_n(d))$ has $\binom{d+1}{2}$ edges, $\overline{G(U_n(d))}$ must have $\binom{n}{2} - \binom{d+1}{2} = L_n((n-1)-d)$ edges.

As for the graph degeneracy, we know that $K_{n-(d+1)}$ must be a subgraph of $\overline{G(U_n(d))}$ from the last equality of (3.5). Since for any a and b we have $K_a \cup K_b = K_{a+b}$ by definition of the complete graph, and

$$K_1 \cup K_{n-(d+1)} \leq \overline{K_{d+1}} \cup K_{n-(d+1)} = \overline{G(U_n(d))}$$

by its construction, we have that $K_{n-(d+1)+1}$ is a subgraph of $\overline{G(U_n(d))}$, and therefore $\text{degen}(\overline{G(U_n(d))}) \geq n-(d+1)$. However, $\text{degen}(\overline{G(U_n(d))}) < n-(d+1)+1$ because $\overline{G(U_n(d))} \setminus K_{n-(d+1)}$ contains no edges; i.e., it is impossible for $K_{n-(d+1)+m}$ to be a subgraph of $\overline{G(U_n(d))}$ for any $m > 1$, so $\text{degen}(\overline{G(U_n(d))}) = n-(d+1)$, as desired.

4 Asymptotics of the ED model polytope and its normal fan

Since we will let $n \to \infty$ in this section, it will be necessary to rescale the polytope $\mathcal{P}_n$ so that it is contained in $[0,1]^2$, for each n. Thus, we divide the graph degeneracy parameter by $n-1$ and the edge parameter by $\binom{n}{2}$, as we have already done in (2.1).

While this rescaling has little impact on shape of $\mathcal{P}_n$ discussed in Section 3, it does affect its normal fan, a key geometric object that plays a crucial role in our subsequent analysis. We describe the normal fan of the normalized polytope next.

PROPOSITION 4.1. *All of the perpendicular directions to the faces of $\mathcal{P}_n$ are:*

$$\{\pm(1,-m) : m \in \{1,2,\ldots,n-1\}\}.$$

So, after normalizing, we get that the directions are

$$\{\pm(1,-\frac{2}{\alpha n}) : \alpha \in \{1/(n-1), 2/(n-1), \ldots, 1\}\}.$$

Proof. The slopes of the line segments defining each face of $\mathcal{P}_n$ are $1/\Delta_U(d) = 1/d$ in the unnormalized parametrization. To get the slopes of the normalized polytope, just multiply each slope by $\binom{n}{2}/(n-1) = n/2$.

Our next goal is to describe the limiting shape of the normalized model polytope and its normal fan as $n \to \infty$. We first collect some simple facts about the limiting behavior of the normalized graph degeneracy and edge count.

PROPOSITION 4.2. *If $\alpha \in [0,1]$ such that $\alpha(n-1) \in \mathbb{N}$ (so that α parameterizes the normalized graph degeneracy), then*

$$\lim_{n\to\infty} \frac{U_n(\alpha(n-1))}{\binom{n}{2}} = \alpha^2.$$

Furthermore, due to the rotational symmetry of $\mathcal{P}_n$,

$$\lim_{n\to\infty} \frac{L_n(\alpha(n-1))}{\binom{n}{2}} = 1-(1-\alpha)^2.$$

Proof. By definition, $U_n(\alpha(n-1)) = \frac{\alpha^2}{2}n^2 + o(n)$. Hence,

$$\frac{U_n(\alpha(n-1))}{\binom{n}{2}} = \frac{\frac{\alpha^2}{2}n^2 + o(n)}{\frac{n(n-1)}{2}} \to \alpha^2.$$

We can now proceed to describe the set limit corresponding to the sequence $\{\mathcal{P}_n\}_n$ of model polytopes. Let

$$P = \mathrm{cl}\left\{t \in \mathbb{R}^2 : t = t(G), G \in \mathcal{G}_n, n = 1,2,\ldots\right\}$$

be the closure of the set of all possible realizable statistics (2.1) from the model. Using Propositions 4.1 and 4.2 we can characterize P as follows.

LEMMA 4.1. *1. $\mathcal{P}_n \subset \mathcal{P}$ for all n and $\lim_n \mathcal{P}_n = \mathcal{P}$.*

2. Let L and U be functions from $[0,1]$ into $[0,1]$ given by

$$L(x) = 1-\sqrt{1-x} \quad \text{and} \quad U(x) = \sqrt{x}.$$

Then,

$$\mathcal{P} = \left\{(x,y) \in [0,1]^2 : L(x) \le y \le U(x)\right\}.$$

The convex set $\mathcal{P}$ is depicted at the top of Figure 4.

In order to study the asymptotics of extremal properties of the ED model, the final step is to describe all the normals to $\mathcal{P}$. As we will see in the next section, these normals will correspond to different extremal behaviors of the model. Towards this end, we define the following (closed, pointed) cones

$$\begin{aligned} C_\emptyset &= \mathrm{cone}\left\{(1,-2),(-1,0)\right\}, \\ C_{\mathrm{complete}} &= \mathrm{cone}\left\{(1,0),(-1,2)\right\}, \\ C_{\mathrm{U}} &= \mathrm{cone}\left\{(-1,0),(-1,2)\right\}, \\ C_{\mathrm{L}} &= \mathrm{cone}\left\{(1,0),(1,-2)\right\}, \end{aligned}$$

where, for $A \subset \mathbb{R}^2$, $\mathrm{cone}(A)$ denote the set of all conic (non-negative) combinations of the elements in A. It is clear that $C_\emptyset$ and C_{complete} are the normal fan to the points $(0,0)$ and $(1,1)$ of $\mathcal{P}$. As for the other two cones, it is not hard to see that the set of all normal rays to the edges of the upper, resp. lower, boundary of $\mathcal{P}_n$ for all n are dense in C_U, resp. C_L. As we will show in the next section, the regions $C_\emptyset$ and C_{complete} indicate directions that, for large n, of statistical degeneracy towards the empty and complete graphs, respectively. On the other hand C_U and C_L contain directions of non-trivial convergence to extremal configurations of maximal and minimal graph degeneracy. See Figure 3 and the middle and lower part of Figure 4.

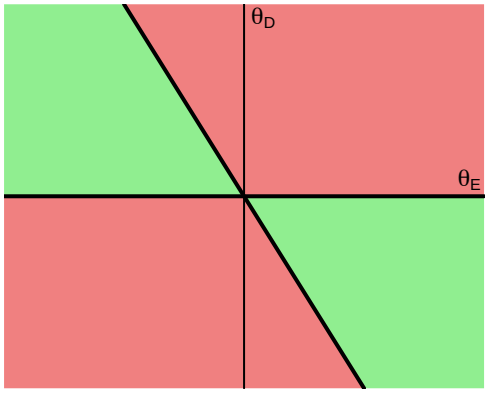


Figure 3: The green regions indicate directions of nontrivial convergence. The bottom-left and top-right red regions indicate directions towards the empty graph $\bar{K}_n$ and complete graph K_n, respectively.

5 Asymptotical Extremal Properties of the ED Model

In this section we will be describe the behavior of distributions from the ED model of the form $P_{n,\beta+rd}$, where d is a non-zero point in $\mathbb{R}^2$ and r a positive number. In particular, we will consider the case in which d and β are fixed, but n and r are large (especially r). We will show that there are four possible types of extremal behavior of the model, depending on d, the "direction" along which the distribution becomes

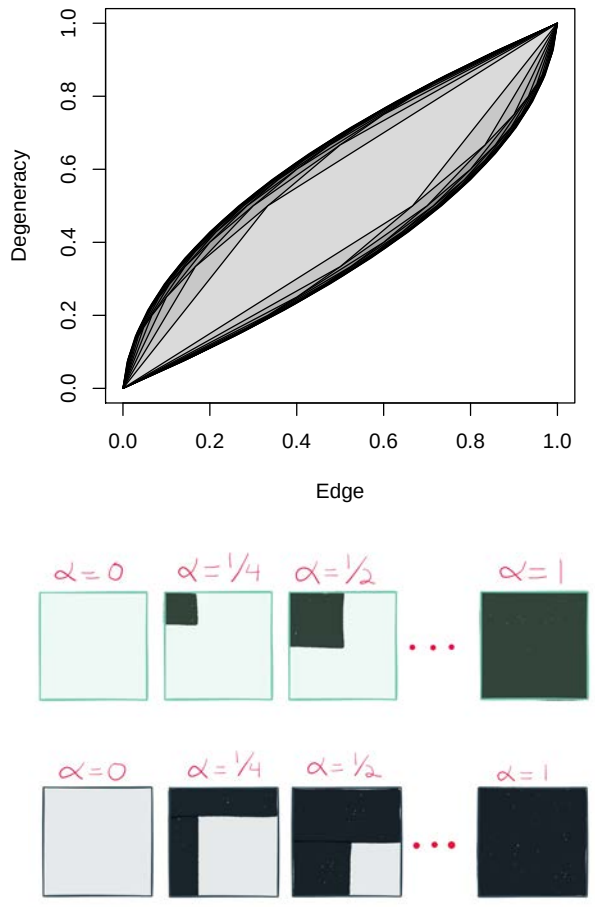


Figure 4: (Top) the sequence of normalized polytopes $\{\mathcal{P}_n\}_n$ converges outwards, starting from $\mathcal{P}_3$ in the center. (Middle) and (bottom) are the representative infinite graphs along points on the upper and lower boundaries of $\mathcal{P}$, respectively, depicted as graphons [15] for convenience.

extremal (for fixed n and as r grows unbounded). This dependence can be loosely expressed as follows: each d will identify one and only one value $\alpha(d)$ of the normalized edge-degeneracy pairs such that, for all n and r large enough, $P_{n,\beta+rd}$ will concentrate only on graphs whose normalized edge-degeneracy value is arbitrarily close to $\alpha(d)$.

In order to state this result precisely, we will first need to make some elementary, yet crucial, geometric observations. Any $d \in \mathbb{R}^2$ defines a normal direction to one point on the boundary of $\mathcal{P}$. Therefore, each $d \in \mathbb{R}^2$ identifies one point on the boundary of $\mathcal{P}$, which we will denote $\alpha(d)$. Specifically, any $d \in C_\emptyset$ is in the normal cone to the point $(0,0) \in \mathcal{P}$, so that $\alpha(d) = (0,0)$ (the normalized edge-degeneracy of the empty graph) for all $d \in C_\emptyset$ (and those points only). Similarly, any $d \in C_{\text{complete}}$ is in the normal cone to the point $(1,1) \in \mathcal{P}$, and therefore $\alpha(d) = (1,1)$ (the normalized edge-degeneracy of K_n) for all $d \in C_{\text{complete}}$ (and those points only). On the other hand, if $d \in \text{int}(C_L)$, then d is normal to one point on the upper boundary of $\mathcal{P}$. Assuming without loss of generality that $d = (1, a)$, $\alpha(d)$ is the point (x, y) along the curve $\{L(x), x \in [0,1]\}$ such that $L'(x) = -\frac{1}{a}$. Notice that, unlike the previous cases, if d and d' are distinct points in $\text{int}(C_L)$ that are not collinear, $\alpha(d) \neq \alpha(d')$. Analogous considerations hold for the points $d \in \text{int}(C_U)$: non-collinear points map to different points along the curve $\{U(x), x \in [0,1]\}$.

With these considerations in mind, we now present our main result about the asymptotics of extremal properties of the ED model.

THEOREM 5.1. *Let $d \neq 0$ and consider the following cases.*

- $d \in \text{int}(C_\emptyset)$.
  Then, for any $\beta \in \mathbb{R}^2$ and arbitrarily small $\epsilon \in (0,1)$ there exists a $n(\epsilon)$ such that for all $n \geq n(\epsilon)$ there exists a $r = r(\epsilon, n)$ such that, for all $r \geq r(\epsilon, n)$ the empty graph has probability at least $1-\epsilon$ under $P_{n,\beta+rd}$.
- $d \in \text{int}(C_{\text{complete}})$.
  Then, for any $\beta \in \mathbb{R}^2$ and arbitrarily small $\epsilon \in (0,1)$ there exists a $n(\epsilon)$ such that for all $n \geq n(\epsilon)$ there exists a $r = r(\epsilon, n)$ such that, for all $r \geq r(\epsilon, n)$ the complete graph has probability at least $1-\epsilon$ under $P_{n,\beta+rd}$.
- $d \in \text{int}(C_L)$.
  Then, for any $\beta \in \mathbb{R}^2$ and arbitrarily small $\epsilon, \eta \in (0,1)$ there exists a $n(\epsilon)$ such that for all $n \geq n(\epsilon, \eta)$ there exists a $r = r(\epsilon, n)$ such that, for all $r \geq r(\epsilon, \eta, n)$ the set of graphs in $\mathcal{G}_n$ whose normalized edge-degeneracy is within η of $\alpha(d)$ has probability at least $1-\epsilon$ under $P_{n,\beta+rd}$.
- $d \in \text{int}(C_U)$.
  Then, for any $\beta \in \mathbb{R}^2$ and arbitrarily small $\epsilon, \eta \in (0,1)$ there exists a $n(\epsilon)$ such that for all $n \geq n(\epsilon, \eta)$ there exists a $r = r(\epsilon, n)$ such that, for all $r \geq r(\epsilon, \eta, n)$ the set of graphs in $\mathcal{G}_n$ whose normalized edge-degeneracy is within η of $\alpha(d)$ has probability at least $1-\epsilon$ under $P_{n,\beta+rd}$.

Remarks. We point out that the directions along the boundaries of $C_\emptyset$ and C_{complete} are not part of our

results. Our analysis can also accommodate those cases, but in the interest of space, we omit the results. More importantly, the value of β does not play a role in the limiting behavior we describe. We further remark that it is possible to formulate a version of Theorem 5.1 for each finite n, so that only r varies. In that case, by Proposition 4.1, for each n there will only be $2(n-1)$ possible extremal configurations, aside from the empty and fully connected graphs. We have chosen instead to let n vary, so that we could capture all possible cases.

Proof. We only sketch the proof for the case $d \in \text{int}(C_L)$, which follows easily from the arguments in [22], in particular Propositions 7.2, 7.3 and Corollary 7.4. The proofs of the other cases are analogous. First, we observe that the assumption (A1)-(A4) from [19] hold for the ED model. Next, let n be large enough such that d is not in the normal cone corresponding to the points $(0,0)$ and $(1,1)$ of $\mathcal{P}_n$. Then, for each such n, d either defines a direction corresponding to the normal of an edge, say e_n, of the upper boundary of $\mathcal{P}_n$ or d is in the interior of the normal cone to a vertex, say v_n, of $\mathcal{P}_n$. Since $\mathcal{P}_n \to \mathcal{P}$, n can be chosen large enough so that either the vertices of e_n or v_n (depending on which one of the two cases we are facing) are within η of $\alpha(d)$.

Let us first consider the case when d is normal to the edge v_n of $\mathcal{P}_n$. Since every edge of $\mathcal{P}_n$ contains only two realizable pairs of normalized edge count and graph degeneracy, namely its endpoints, using the result in [22], one can choose $r = r(n, \epsilon, \eta)$ large enough so that at least $1-\epsilon$ of the mass probability of $P_{n,\beta+rd}$ concentrates on the graphs in $\mathcal{G}_n$ whose normalized edge-degeneracy vector is either one of the two vertices in e_n. The claim follows from the fact that these vertices are within η of $\alpha(q)$ For the other case in which d is in the interior of the normal cone to the vertex v_n, again the results in [22] yield that one can can choose $r = r(n, \epsilon, \eta)$ large enough so that at least $1-\epsilon$ of the mass probability of $P_{n,\beta+rd}$ concentrates on graphs in $\mathcal{G}_k$ whose normalized edge-degeneracy vector is v_n. Since v_n is within η of $\alpha(q)$ we are done.

The interpretation of Theorem 5.1 is as follows. If d is a non-zero direction in $C_\emptyset$ and C_{complete}, then $P_{n,\beta+rd}$ will exhibit statistical degeneracy regardless of β and for large enough r, in the sense that it will concentrate on the empty and fully connected graphs, respectively. As shown in Figure 3, $C_\emptyset$ and C_{complete} are fairly large regions, so that one may in fact expect statistical degeneracy to occur prominently when the model parameters have the same sign. The extremal directions in C_U and C_L yields instead non-trivial behavior for large n and r. In this case, $P_{n,\beta+rd}$ will concentrate on graph configurations that are extremal in sense of exhibiting nearly maximal or minimal graph degeneracy given the number of edges.

Taken together, these results suggest that care is needed when fitting the ED model, as statistical degeneracy appears to be likely.

6 Discussion

The goal of this paper is to introduce a new ERGM and demonstrate its statistical properties and asymptotic behavior captured by its geometry. The ED model is based on two graph statistics that are not commonly used jointly and capture complementary information about the network: the number of edges and the graph degeneracy. The latter is extracted from important information about the network's connectivity structure called cores and is often used as a descriptive statistic.

The exponential family framework provides a beautiful connection between the model geometry and its statistical behavior. To that end, we completely characterized the model polytope in Section 3 for finite graphs and Section 4 for the limiting case as $n \to \infty$. The most obvious implication of the structure of the ED model polytope is that the MLE exists almost surely for large graphs. Another is that it simplifies greatly the problem of projecting noisy data onto the polytope and finding the nearest realizable point, as one need only worry about the projection. Such projections play a critical role in data privacy problems, as they are used in computing a private estimator of the released data with good statistical properties; see [9]. In fact, our geometric results imply that [11, Problem 4.5] is easier for the ED model than the beta model based on node degrees, which was solved in [8]. Finally, the structure of the polytope and its normal fan reveal various extremal behaviors of the model, discussed in Section 5.

Note that the two statistics in the ED model summarize very different properties of the observed graph, giving this seemingly simple model some expressive power and flexibility. In graph-theoretic terms, the degeneracy summarizes the core structure of the graph, within which there can be few or many edges (see [11] for details); combining it with the number of edges produces Erdős-Renyi as a submodel.

As discussed in Section 2, different choices of the parameter vector, that is, values of the edge-degeneracy pair, lead to rather different distributions, from sparse to dense graphs as both parameters are negative or positive, respectively, as well as graphs where edge count and degeneracy are balanced for mixed-sign parameter vectors. Our results in Section 5 provide a catalogue of such behaviors in extremal cases and for large n. The asymptotic properties we derive offer interesting insights on the extremal asymptotic behavior of the

ED model. However, the asymptotic properties of non-extremal cases, that is, those of distributions of the form $P_{n,\beta}$ for *fixed* β and diverging n, remain completely unknown. While this an exceedingly common issue with ERGMs, whose asymptotics are extremely difficult to describe, it would nonetheless be desirable to gain a better understanding of the ED model when the network is large. In this regard, the variation approach put forward by [6], which provides a way to resolve the asymptotics of ERGMs in general, may be an interesting direction to pursue in future work.

References

[1] J. Bae and S. Kim, *Identifying and ranking influential spreaders in complex networks by neighborhood coreness*, Physica A: Statistical Mechanics and its Applications, 395 (2014), pp. 549-559.

[2] O. Barndorff-Nielsen, *Information and exponential families in statistical theory*, Wiley (1978).

[3] A. Bickle, *The k-cores of a graph*, Ph.D. dissertation, Western Michigan University (2010).

[4] L. D. Brown, *Fundamentals of statistical exponential families*, IMS Lecture Notes, Monograph Series 9 (1986).

[5] S. Carmi and S. Havlin and S. Kirkpatrick and Y. Shavitt and E. Shir,*A model of Internet topology using k-shell decomposition*, Proceedings of the National Academy of Sciences, 104(27) (2007), pp. 11150-11154.

[6] S. Chatterjee and P. Diaconis, *Estimating and understanding exponential random graph models*, Annals of Statistics, 41(5) (2013), pp. 2428–2461.

[7] S. E. Fienberg and A. Rinaldo, *Maximum likelihood estimation in log-linear models*, Annals of Statistics, 40(2) (2012).

[8] A. Goldenberg and A. X. Zheng and S. E. Fienberg and E. M. Airoldi, *A survey of statistical network models*, Foundations and Trends in Machine Learning 2(2) (2009), pp. 129–233.

[9] S. M. Goodreau, *Advances in exponential random graph (p^*) models applied to a large social network*, Special Section: Advances in Exponential Random Graph (p^*) Models, Social Networks 29(2) (2007), pp. 231-248.

[10] M. S. Handcock, *Assessing degeneracy in statistical models of social networks*, working paper no. 39, Center for Statistics and the Social Sciences, University of Washington (2003).

[11] V. Karwa and M. J. Pelsmajer and S. Petrović and D. Stasi and D. Wilburne, *Statistical models for cores decomposition of an undirected random graph*, preprint arXiv:1410.7357 (v2) (2015).

[12] V. Karwa and A. Slavković, *Inference using noisy degrees: differentially private β-model and synthetic graphs*, Annals of Statistics 44(1) (2016), pp. 87–112.

[13] V. Karwa and A. Slavković and P. Krivitsky, *Differentially private exponential random graphs*, Privacy in Statistical Databases, Lecture Notes in Computer Science, Springer (2014), pp. 142-155.

[14] M. Kitsak and L.K. Gallos and S. Kirkpatrick and S. Havlin and F. Liljeros and L. Muchnik and H.E. Stanley and H. Makse, *Identification of influential spreaders in complex networks*, Nature Physics, 6(11) (2010), pp. 880–893.

[15] L. Lovász, *Large networks and graph limits*, American Mathematical Society (2012).

[16] S. Pei and L. Muchnik and J. Andrade Jr. and Z. Zheng and H. Maske, *Searching for superspreaders if information in real-world social media*, Nature Scientific Reports, (2012).

[17] S. Petrović, *A survey of discrete methods in (algebraic) statistics for networks*, Proceedings of the AMS Special Session on Algebraic and Geometric Methods in Discrete Mathematics, Heather Harrington, Mohamed Omar, and Matthew Wright (editors), Contemporary Mathematics (CONM) book series, American Mathematical Society, to appear.

[18] A. Razborov, *On the minimal density of triangles in graphs*, Combin. Probab. Comput. 17 (2008), pp. 603–618.

[19] A. Rinaldo and S. E. Fienberg and Y. Zhou, *On the geometry of discrete exponential families with application to exponential random graph models*, Electronic Journal of Statistics, 3 (2009), pp. 446–484.

[20] A. Rinaldo and S. Petrović and S. E. Fienberg, *Maximum likelihood estimation in the β-model*, Annals of Statistics 41(3) (2013), pp. 1085–1110.

[21] C. R. Shalizi and A. Rinaldo, *Consistency under sampling of exponential random graph models*, Annals of Statistics, 41(2) (2013), pp. 508–535.

[22] M. Yin and A. Rinaldo and S. Fadnavis, *Asymptotic quantization of exponential random graphs*, to appear in the Annals of Applied Probability (2016).

Collaborative SVM Classification in Skewed Peer-to-Peer Networks

Umer Khan* Alexandros Nanopoulos† Lars Schmidt-Thieme*

Abstract

Mining patterns from large-scale networks, such as Peer-to-Peer (P2P), is a challenging task, because centralization of data is not feasible. The goal is to develop distributed mining algorithms that are communication efficient, scalable, asynchronous, robust to peer dynamism, and which achieve accuracy as close as possible to centralized ones. In this paper, we present a collaborative classification method based on Support Vector Machines in large scale P2P networks. Most of the widely known P2P systems are prone to Data or Execution skew either due to emergence of small-world network topologies in their overlays, or hierarchical data placements requirements. In such an inherently skewed distribution of node links and eventually data, existing approaches of sharing local machine learning models only with immediate neighbors may deprive a vast majority of scarcely connected peers of knowledge that can improve their local prediction accuracies. The main idea of the proposed approach is to allow communication beyond the direct neighbors in a controlled way that improves the classification accuracy and, at the same time, keeps the communication cost low throughout the network. Besides using benchmark Machine Learning datasets for extensive experimental evaluations, we have evaluated the proposed method particularly for music genre classification to exhibit its performance in a real application scenario. Our collaborative classification method outperforms baseline approaches of model propagation by improving the overall classification performance substantially at the cost of a tolerable increase in communication.

Keywords: Distributed, Classification, P2P, SVM, Skewed

1 Introduction

1.1 Motivation: In recent years there is an increasing interest for analytical methods that perform data mining over large-scale data distributed over the peers of P2P networks such as eDonkey, BitTorrent and Gnutella to support applications like distributed document classification for automatic structuring of heterogeneous collections and Web directories, distributed recommendation systems, distributed intrusion detection, personalized media retrieval, online matchmaking, distributed spam classification, and many others. Data Mining in large-scale distributed P2P networks is a challenging task, because centralization of data is not feasible. In P2P networks, computing devices might be connected to the network temporarily, communication is unreliable and possibly with limited bandwidth, resources of data and computation can be distributed sparsely, and the data collections are evolving dynamically. A scheme which centralizes the data stored all over a P2P network is usually not feasible, because any change must be reported to the central peer (server), as it might alter the result significantly. Therefore, the goal is to develop distributed mining algorithms that are communication efficient, scalable, asynchronous, and robust to peer dynamism, which achieve accuracy as close as possible to centralized ones.

Existing research in data mining for P2P networks focuses on developing local classification or clustering algorithms which use primitive operations, such as distributed averaging, majority voting and other aggregate functions to combine results of local classifiers in order to establish a consensus of classification decisions among peers in a local neighborhood of a P2P network [16]. [6] proposed a distributed decision tree for P2P networks, by using distributed majority voting. Most of these locally synchronized algorithms are reactive in a sense that they tend to monitor every change in data and keep track of data statistics. Recently researchers have proposed methods to learn Support Vector Machines(SVM) classifier in Peer-to-Peer networks, based on the idea of exchanging local SVM models with the immediate neighbors [18],[3]

One inherent problem of the aforementioned approaches is that they do not consider P2P networks with *scale-free* topology where the distribution of peer links and the amount of data stored in peers follows the power-law distribution, i.e. $d_p \sim x^{-a}$ where d_p refers to degree of a peer. In particular, such networks are prone to *Execution skew* and *Data skew*. Execution skew refers to non-uniform data access across the partitions of peers, while the Data skew describes an uneven distribution of data across these partitions. Most of the widely known real-world P2P systems like FreeNet, Gnutella, BitTorrent, eDonkey, FastTrack, Gia, UMM,

*Information Systems and Machine Learning Lab, University of Hildesheim, Germany, {khan, schmidt-thieme}@ismll.de

†Data Scientist at a Private Organization, alexandros.nanopoulos@gmail.com

Phenix are prone to data or execution skew due to the scale-free growth of network topologies in their unstructured overlays [17], [2]. Another reason of occurrence of such skews are the hierarchical peer and data placements rules or tendencies in most of these P2P networks, basically for efficient query routing, scalability, handling node dynamicity and fault-tolerance [13]. Particularly, this is the case with hierarchical P2P overlays with multiple levels categorizing the peers according to their heterogeneous routing roles and varying loads [12]. In such widely used network topologies a group of few *super peers* from top levels serve a vast majority of regular or weak peers in the lower levels through routes spanning multiple levels and several hops. As a result, there are usually very few peers with many connections and large amount of data, whereas a vast majority of peers have few connections and very small amount of local data. Additionally, it has been demonstrated that the degree of a peer tends to correlate with the amount of data this peer contains [19],[20]. This correlation can be explained with the fact that in most hierarchical P2P networks like KaZaA and Gnutella, peers tend to connect with super-peers which offer faster response to data queries with high bandwidth, and show high availability. For example, in P2P song sharing, songs which are very popular are shared by many peers, but only very few peers share the vast majority of songs [11]. Since the probability of finding a specific song is much higher with these super-peers, they attract a lot of connections from regular peers. On the other hand, a vast majority of peers are *free riders* [9], i.e., they share almost no or very little data, and hence are allowed fewer connections. As a consequence, such regular peers usually do not have a sufficiently large local training data set and are unable to generate accurate classification models.

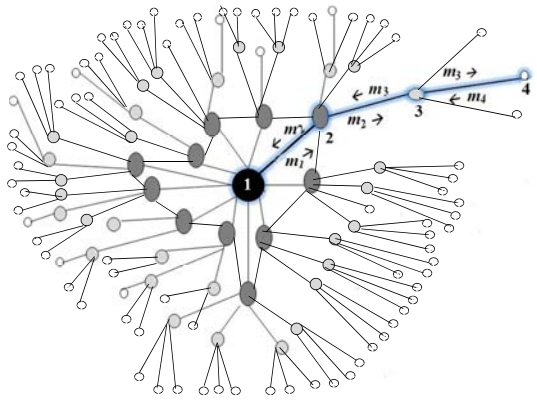


Figure 1: Conceptual example of P2P network with power-law topology and model exchange

The aforementioned idea is exemplified in Figure 1, which illustrates transitive model propagation in a P2P network topology that is 'scale free'. Consider the shadowed path in Figure 1, which contain peers $\{1 \dots 4\}$. Peer 1 is a super-peer, peer 2 has intermediate degree, whereas peers 3 and 4 are weak peers (in Figure 1, peers with higher degree are depicted darker and bigger and those with lower degree are lighter in color and smaller). With a direct propagation scheme, each peer p exchanges its local model m_p with all its neighbors. Peer 1 shares its model m_1 with peer 2, which allows peer 2 to improve its local model m_2, because peer 1 is a super-peer and, thus model m_1 is informative. However, the weak peers, such as peer 4, would not benefit from this, because it exchanges models only with peer 3, which is also weak. Therefore, although a communication cost is paid to exchange models among weak peers, there will be no improvement in the accuracy for those weak peers that are not connected to a super-peer. This problem becomes more severe, as the local models of weak peers may be originally (before any model exchange) very inaccurate and thus classification using these models will be meaningless (i.e., accuracy is close to random classification). As a result, weak peers that comprise the vast majority of a P2P network, may not be able to perform any meaningful classification. By allowing transitive propagation of models, local models of super-peers (such as peer 1) will be able to reach weak peers (such as peer 4) connected to them through paths with length larger than one. Based on these findings, the above mentioned approaches for model propagation [3], [18] may not scale well in inherently scale-free P2P networks as exchanging models only with immediate neighbors restricts the informative knowledge from the few high level *super-peers* with large number of peer connections and data collections, to reach the vast majority of low level regular or *weak* peers separated by multiple hops. With very few connections and not enough discriminative patterns, large number of these regular peers are unable to learn effective classifiers despite the cost of communication for model exchange among these peers as shown in section 4.3.

1.2 Contribution and Outline: The focus of this paper is a collaborative classification method based on Support Vector Machines in the decentralized hierarchical P2P networks considering the above mentioned topological characteristics. We chose SVM as for many datasets it can perform favorably against other classifiers. Moreover, it is the most promising classifier based on convex quadratic programming in which global optimal solutions can be obtained. We propose a transitive model exchange method (TRedSVM) to learn SVM classifier collaboratively among the peers in the

global P2P network topology. In contrast to existing approaches [3], [18], the proposed approach takes into account the scale-free topology of P2P networks and allows communication in a transitive way, i.e., *beyond* immediate neighbors. This way, our method helps not so well connected peers to receive local models of other, better connected peers, in order to enhance the local models of the former, and thus to substantially improve classification accuracy over the entire P2P network. Since uncontrolled transitive propagation of classification models can increase the communication cost (leading eventually to flooding the network), TRedSVM keeps the communication in control first by using a reduced variant of SVM called Reduced-SVM (RSVM) [15] to learn a local classification model at each peer and secondly but more importantly by performing a weighted reduction of model sizes at each transitive step. Therefore, TRedSVM is able to transitively propagate most significant and compact form of knowledge resulting in substantial increase in classification accuracy of the overall network while keeping the communication cost low.

In order to assess the efficiency of TRedSVM in real applications, we have applied it to music genre classification problem besides using benchmark Machine Learning datasets for comparison with baseline model propagation schemes and data centralization approaches. As modern P2P systems rely on data replication to maintain high data availability and faster response time, we have evaluated the proposed method with data replication based on relative popularity of data items. Our experimental results show that TRedSVM improves the overall classification accuracy of the network substantially at the cost of tolerable increase in communication. It also provides a reasonable trade-off between communication cost and overall prediction accuracy, which allows it to compare favorably against existing baseline approaches.

The rest of the this paper is organized as follows: Section 2 describes related work. Section 3 describes the proposed approach of TRedSVM, followed by experiments and evaluations in Section 4 and conclusions in the end.

2 Related Work

Current *state-of-the-art* in P2P data mining has evolved from earlier attempts to perform some primitive operations (average, sum, max, random sampling) in distributed setting. For example, [10] proposed a gossip-based randomized algorithm to compute aggregates (MAX) in an n-node overlay network. [14] presented distributed algorithms for effectively calculating basic statistics of data using the newscast model of computation. A main goal of these works is to lay a foundation for more sophisticated data mining algorithms and applications.

Recent approaches in P2P data mining focus on developing local classification or clustering algorithms which in turn make use of primitive operations such as distributed averaging, majority voting and other aggregate functions. Most representative work in this regard is distributed decision tree induction [6], distributed K-Means clustering [8] and distributed classification [16]. A common aspect of these approaches is that they make use of *local* algorithms, which compute their result using information from just a handful of nearby neighbors. A major limitation of such *local* algorithms is that up till now, they have been conceived only to support primitive aggregate operations. A more well known *local* algorithm is Majority Voting by [21]. In this algorithm, each peer maintains its local belief of an estimate of a global sum based on the number of nodes in that network, to confirm whether a majority threshold is met. If not then this peer needs to send messages to its neighbors to re-estimate the global sum. A similar work is published by [16] to achieve a consensus among neighboring peers using distributed plurality voting. Most of these locally synchronized algorithms are reactive in a sense that they tend to monitor every single change in data and keep track of data statistics in their local neighborhood, which also requires extra polling messages for coordination.

Based on model propagation, recently another important work [18] proposed collaborative SVM model exchange for distributed classification in P2P networks. Local SVM models are learned on each peer and exchanged with immediate neighbors. All the received models are merged to build a final classifier. A similar method is proposed in [3] by cascading SVM models among immediate neighbors in a communication efficient way. Despite the several suitable characteristics of such methods, their major limitation is that they consider uniform and controlled placement of peers and data in P2P networks, and exchange local SVM models only with immediate neighbors. As discussed in section-1, these approaches may not scale well in most real-world P2P networks which follow a hierarchical network topology with highly skewed distribution of peer degrees.

3 Methodology: Transitive Reduced-SVM

3.1 Basic Concepts and Notations: More formally, we consider an ad-hoc P2P network comprising of a set of such k autonomous peers $P = \{1, 2, \ldots, k\}$. The topology of the P2P network is represented by a (connected) graph $G(P, E)$, in which each peer $p \in P$ is

represented by a vertex and an edge $\{p,q\} \in E$, where $E \subseteq \{\{p,q\} : p,q \in P\}$, whenever peer p is connected to peer q. Also let N_p denote the set of immediate neighbors of p, i.e., $N_p = \{q \in P | q \neq p, \{p,q\} \in E\}$. The local training data set on a peer p is denoted as $X_p \subseteq \mathbb{R}^{d+1}$, where d is the number of data features. Based on the local training data set X_p, each peer p can first build its local classification model m_p. Since exchanging classification models results in additional communication cost, what is required is the construction of local classification models that are both accurate and compact, i.e., they can be represented with a small amount of information that is needed to be transmitted between neighboring peers. To attain this, we have used a variant of SVM called Reduced-SVM (RSVM) [15]. Compared to SVM, RSVM has been reported to be able to reduce the number of generated support vectors significantly while maintaining very good classification accuracy. A brief description of RSVM is given in next section. Interested readers are referred to [15] for details.

3.2 Building Local Classifier: As the initialization step of the proposed method, RSVM classification model is built using the local training data at each peer. For the convenience of notation, here we represent local training data with X instead of X_p. Given is a training set of instance-label pairs $\{(x_j, y_j)\}_{j=1}^{|X|}$, where $x_j \in \mathbb{R}^d$ is an input vector and $y_j \in \{-1, 1\}$ is the corresponding class label. RSVM starts by adding the term $\frac{\gamma^2}{2}$ to the SVM objective function, and solves the following optimization problem:

$$\min_{(w,\gamma,\varepsilon)} \quad \frac{1}{2}\left(\|w\|_2^2 + \gamma^2\right) + \frac{C}{2}\|\varepsilon\|_2^2 \tag{3.1}$$

$$\text{subject to} \quad D\left(K\left(X, X^T\right) w - \mathbf{1}\gamma\right) \geq \mathbf{1} - \varepsilon \tag{3.2}$$

where $C > 0$ balances training error and the *regularization term* in the objective function. The weight vector $w = \left(w_1, w_2, \ldots, w_{|X|}\right)$ is perpendicular to the hyperplane that separates the two classes, and γ is the intercept term. D is an $|X| \times |X|$ diagonal matrix such that $D_{jj} = y_j$, to specify the target class of each instance. K is the kernel function which computes the *dot product* between input vectors in a mapping feature space. $K\left(X, X^T\right)$ is a $|X| \times |X|$ matrix such that $K\left(x_i, x_j^T\right) \equiv \phi\left(x_i\right)^T \phi\left(x_j\right)$ where $\phi(x)$ maps x into a higher (may be infinite) dimensional space. Solving this optimization for massive data sets by computing the full dense non-linear kernel matrix in (3.2), results in unwieldy storage and computation overhead. To avoid this overhead, the RSVM replaces the full kernel matrix $K\left(X, X^T\right)$ with the reduced one $K\left(X, \tilde{X}^T\right) \in \mathbb{R}^{|X| \times r}$, where $\tilde{X}$ consists of r random instances from X, such that $r \ll |X|$ where $|X|$ being the number of instances in X. Subsequently RSVM classifier is learned by solving the following unconstrained minimization.

$$\min_{(\tilde{w},\gamma) \in \mathbb{R}^{r+1}} \quad \frac{1}{2}\left(\|\tilde{w}\|_2^2 + \gamma^2\right) + \frac{C}{2}\| \left(\mathbf{1} - D\left\{K\left(X, \tilde{X^T}\right)\tilde{w} - \mathbf{1}\gamma\right\}\right)_+ \|_2^2 \tag{3.3}$$

Note that the whole problem including coefficients $\tilde{w}$ of separating hyperplane and the size of subset data $\tilde{X}$ used to represent this hyperplane, is reduced to a positive definite linear system of size r. This size can be specified by the *subset-ratio* η of local data X to be used for RSVM such that $r = \lceil |X| \times \eta \rceil$.

3.3 TRedSVM - Transitive Reduced-SVM Approach The pseudo-code of the asynchronous TRedSVM algorithm is given in Algorithm 1 and a list of symbols used is provided in Table 1. TRedSVM starts by learning local RSVM model m_p at each peer p in the first phase, using a predefined reduction parameter η initialized by the network for all the peers globally. Subset-ratio η directly affects the size of the resulting model m_p, since it selects the random subset of instances $\tilde{X}_p$, used to learn a nonlinear hyperplane represented by support vectors, as described in Equation (3.3). A large value of η results in a potentially larger model size, and vice versa. Then, each peer p using its neighbor list N_p propagates the local model m_p to all immediate neighbors.

Each peer p keeps a MAP_p to map a model to its sender and a local buffer REC_p to keep received models. Moreover for receiving models, a peer waits for time t until models m_q from all the neighbors $q \in N_p$ have been received. Once all the neighboring models have been received, in the transitive phase, each peer p reduces each of the received models m_q using RSVM with its local weighted reduction parameter η_p (line 25 of Algorithm 1).[1] This step ensures that the size of each of the received models is further reduced and only the most statistically significant form reach the weak peers. As described in Section 1, there is a strong correlation between degrees of nodes and the size of data they carry, in this work we have chosen the value of η_p weighted by the *hubness level* of peer p. Each peer p is assigned to a hubness level according to its *proxy score* $h(p)$ defined as: $h(p) = \frac{(d_p - \mu_d)}{\sigma_d}$, where d_p is degree of peer p, μ_d and σ_d are mean and standard deviation of degrees of all

[1] Parameters like η, η_p and N_p are network parameters provided to each joining peer by its associated super-peer.

Table 1: **List of symbols in TRedSVM**

Symbol	Meaning
X_p	Local data at peer p
N_p	Neighbor list of Peer p
m_p	Local model of p^{th} peer learned using RSVM
MAP_p	Maps a model to its sender identifier
REC_p	Local buffer to keep received models of any peer $n \mid n \neq p$
η	Subset ratio (global) of data to learn local RSVM models
η_p	Subset ratio (local) of data for transitive reduction at peer p defined by its hubness level

the peers in the network, respectively. Hubness levels result by organizing the proxy scores of all peers in a small number of groups (in our experiments we used 5 hubness levels, described in Table 2 in Section 4.1). Hence in TRedSVM, peers with higher hubness levels such as super-peers perform less reduction in model size in the transitive step, so that they forward a larger chunk of this useful information to their associated weak peers. On the other hand, weak peers with low hubness levels have insufficient data to learn useful models, use smaller η_p to perform larger reduction and propagate much smaller portion of possibly redundant knowledge as there is a high probability that super-peers may already have the data points that a weak peer wants to share. After reducing each of the received models to only most significant support vectors in the second phase, for each of these models in REC_p, peer p queries all its neighbors q where $q \in N_p$ and $q \neq n$. In response, if q does not have the queried model or has one with a size similarity smaller than threshold λ, it acknowledges p to send this model. Otherwise, peer q sends a decline. The optimal value of λ is selected empirically and in our experiments we used $\lambda = 50$. Again, this policy is chosen to ensure that a peer only keeps more informative models that have been routed to it by stronger peers. On receipt, each peer adds the model to its buffer and updates the map to $MAP_p(n)$. This process of reduction based relearning and controlled propagation in the second phase continues at each peer until it has received, from all the neighbors, subsequent more stronger models which it should have obtained. Once the propagation to and receipt from all the neighbors is finished, each peer p merges the instances x_j in all the received models $MAP_p(n) | n \in P$ from $q \in N_p$ to its local training data and learns a final SVM model m_p. Such a model propagation scheme can also account for *Peer Dynamism*, i.e. peers can join and leave the network anytime. Even if a peer goes offline, its model still exists on other peers of the network. This factor allows maintaining high classification accuracy by sharing models of peers which were not present in the network at the same time. Finally, it has to be mentioned that although TRedSVM performs transitive propagation of models, it is still a *local* algorithm, in the sense that each peer performs computation using information provided by its neighboring peers. Therefore, the resources required by TRedSVM are independent of the total number of peers.

Algorithm 1 TRedSVM for Classification at peer p

Require: Local training data X_p, t=time to wait between retraining, N_p=List of neighbors, η = Subset ratio to learn peer's local model, η_p = Subset ratio defined by hubness level to use for transitive reduction
Ensure: Updated model m_p

m_p = RSVM (X_p, η)
$MAP_p := \{(p, m_p)\}$
$REC_p := \emptyset$
▷ Communication and Model update threads execute in Parallel
loop{communication thread}
  event=receive_event()
  if type(event)== OFFER_MODEL **then**
    $(n, size_n)$:= args(event)
    if $|MAP_p(n)| == 0$ or $size_n > |MAP_p(n)| + \lambda$ **then**
      send REQUEST_MODEL(n) to sender(event)
    end if
  else if type(event)== REQUEST_MODEL **then**
    n:=args(event)
    send MODEL_PARAMETERS($n, MAP_p(n)$) to sender(event)
  else if type(event)== MODEL_PARAMETERS **then**
    (n, m):=args(event)
    $REC_p(n)$:=m
  end if
end loop
loop{Model update thread}
  start_time = current_time()
  while *current_time()-start_time*$< t$ **do**
    if exists $n \mid |REC_p(n)| > 0$ **then**
      X_p:=$X_p \setminus MAP_p(n) \cup REC_p(n)$
      $MAP_p(n)$:=RSVM($REC_p(n)$,η_p)
      $REC_p(n) = \emptyset$
      send OFFER_MODEL($n, |MAP_p(n)|$) to all $q \in N_p$
    end if
  end while
  m_p:=RSVM(X_p,η)
  use m_p
end loop

4 Performance Evaluation

In this section, we first describe the experimental setup, after which experimental results are presented, and finally we discuss these results.

4.1 Experimental Setup

Topology and Simulation Environment: To generate a multi-level hierarchical P2P network with *power law* distribution of peers, we used the *Barabasi-Albert* model [4] with parameters like *preferential connectivity* and *incremental growth*, where a joining peer p connects to an existing peer q with a probability given as $P(p,q) = \frac{d_q}{\sum_{k \in V} d_k}$, where d_q is degree of existing node q, V is the set of nodes that have already joined the network, and $\sum_{k \in V} d_k$ is the sum of degrees of all nodes that previously joined the network. For this purpose we used the BRITE, a network topology generation framework developed at Boston University [1]. The result is a topology represented by a weighted graph with edge weights representing communication delays in milliseconds. We evaluated our experiments with varied number of peers ranging from 100 to 500. We did not experiment with more peers, since it would result in unrealistically small sizes of local data at peers and could adversely affect the performance of P2P classification systems. For local computation at each peer and monitoring of message exchange, we have built our own simulator (using Java) that simulates distributed dynamic P2P overlay trees. We have used communication delays on the BRITE network graph as measurement of time. Local computations and message exchange are regulated with change of network state, with respect to a common global clock of a simulator.

Experiments were conducted on a cluster of 41 machines, each with 10 Intel-Xeon 2.4GHz processors, 20GB of Ram, and connected by gigabit Ethernet. For learning local classifiers, we used the LIBSVM ([7]) implementation of RSVM. Moreover, to compare the number of support vectors generated by RSVM with regular SVM, the C-SVC implementation of regular SVM is used. For RSVM, we used the RBF kernel, and for each data set, the hyper-parameters γ and C were empirically found using the model selection tool provided by LIBSVM (*covertype* : $\gamma = 2^{-7}$, $C = 2^{-0.1}$ and $cod-rna : \gamma = 2^{-2}$, $C = 2^{-2}$).

Data Sets: We have used existing benchmark Machine Learning data sets which are among the largest in the corresponding widely used repositories to evaluate the performance of our method in comparison to the stated baselines. In particular, we used the:

- *covertype* (581012×54, 7 classes) data set from UCI repository, and
- *cod-rna*(488565 × 8, 2 classes) data set from LIBSVM repository [7].
- *Million Song Dataset*(1,000,000 × 33, 10 classes) data set from Million Song Dataset website [5]

Data Distribution and Hubness Levels: TRedSVM considers a probabilistic distribution of training data such that size $|X_p|$ of data on a peer p is directly proportional to its degree d_p, such that $|X_p| = |\mathcal{X}|.\frac{d_p}{\sum d}$, where $|\mathcal{X}|$ is the size of original (centralized) data set. Figure 2 depicts the highly skewed distribution of the sizes of local training sets among all peers. Based on the resulting distribution of degrees

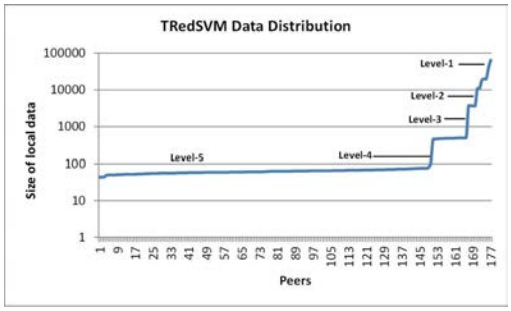


Figure 2: Data distribution among peers in TRedSVM simulation

and data among peers, TRedSVM can consider several hubness levels, as discussed in Section 3.3. In our evaluation we selected five levels ranging from super-peers (Level-1) to weak peers (Level-5). Peers in each hubness level used a corresponding η_p value assigned to their hubness level to learn an RSVM model during the transitive propagation phase. Table 2 lists the η_p values assigned to peers in each hubness level based on their hubness score. Note that the higher the hubness level (closer to Level-1), the higher the η_p value.

Table 2: **Grouping of peers into hubness levels.**

Hubness level	Degree of Peers	Percentage of SVs
Level-1	$d_i \geq 20$	$\eta_p = 0.1$
Level-2	$15 \leq d_i < 20$	$\eta_p = 0.075$
Level-3	$10 \leq d_i < 15$	$\eta_p = 0.05$
Level-4	$5 \leq d_i < 10$	$\eta_p = 0.025$
Level-5	$d_i < 5$	$\eta_p = 0.01$

4.2 Music Genre Classification: Music genre are considered to be important metadata for retrieving songs in the field of music information retrieval. The Million Song Dataset (MSD) is the largest publicly available dataset consisting of audio features and metadata for a million contemporary popular music tracks [5]. But none of the songs in MSD have any genre labels. In order to evaluate TRedSVM in a real application scenario, we have employed it to the task of automatic classification of songs in MSD based on their genres. Artist tags in MSD are used to describe genres and to extract simple features from the tracks of these artists. To select standardized, descriptive and consistent tags, we

have chosen the following 10 tags from the top 50 most popular MusicBrainz tags: classic pop and rock, folk, dance and electronica, jazz and blues, soul and reggae, punk, metal, classical, pop, hip-hop. For features, we have used the following ones from The Echo Nest [5] loudness, tempo, time_signature, key, mode, duration, average and variance of timbre vectors. As described in the following section, TRedSVM significantly improves the percentage of songs with correctly classified genres, as compared to baseline approaches. Although existing approaches in music information retrieval have shown that audio features when combined with lyrics and social tags can help classifying song genre more accurately, to account for the scope of this paper, we have only used simple audio and timbre related features.

4.3 Experimental Results Performance of TRedSVM is evaluated in terms of classification accuracy and communication cost, which are the two most significant aspects for the problem of classification in P2P networks. Comparison is conducted against the state-of-the-art approaches of model propagation among neighboring peers proposed as AllCascade by [3] and CSVM by [18]. In the results below, we refer to these approaches as *Direct Neighbors Exchange*. To illustrate an upper bound on communication cost, we also performed experiments with the network *Flooding* baseline that performs transitive propagation without controlling the communication cost i.e. without performing any reduction on propagated model. Effectiveness of transitive reduction w.r.t Flooding even without querying for redundant model exchange among peers, is also presented as a variant *Uncoordinated-TRedSVM*. Finally, to demonstrate the effectiveness of model exchange, we also consider a baseline method that learns a *Local Model* of classification without any model exchange. In our experiments, we distributed the test data both uniformly across all the peers as well as using the same skewed distribution as the train data. The results from both types of experiments did not show any significant change in the classification accuracy.

Network wide Classification Accuracy: We first focus on average classification accuracy of the whole network. Figure 3 depicts the resulting accuracy for the four examined methods and the three data sets. As expected, the simple baseline of *Local Model* that involves no exchange of models in Figure 3), is clearly outperformed by all other methods. TRedSVM compares favorably to the method that involves model exchange only among direct neighbors. As expected, best accuracy is achieved for naive *Flooding*, but with in-feasible cost of communication as discussed below.

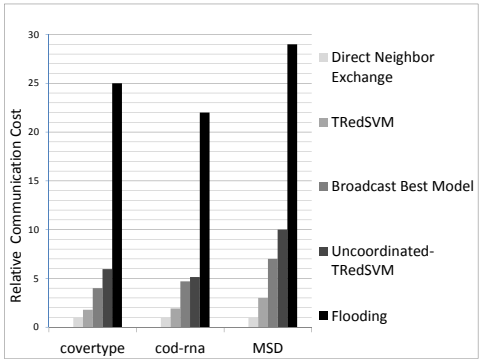


Figure 4: Relative communication cost of TRedSVM compared with the methods of direct exchange and flooding

Communication Cost: Figure 4 illustrates the communication cost incurred by the four examined methods, since the simple baseline that involves no exchange of models has by definition no communication cost. The costs are relative to each other when normalized by cost of *Direct Neighbors Exchange*. Though *Flooding* achieved best accuracy, Figure 4 clearly shows that flooding requires a prohibitive communication cost, when data becomes huge. Thus, it cannot comprise a feasible method in real P2P networks. TRedSVM requires communication cost that is substantially less than that of flooding and only marginally larger compared to the method that involves exchange of models only between direct neighbors. The cost incurred by *Uncoordinated-TRedSVM* demonstrate that transitive reduction, even without coordination, does actually reduces the cost substantially as compared to simple flooding. We have also considered to evaluate a possible scenario in which we broadcast the most accurate model (possibly from super-peers) to all the peers in the network. For example, for *covertype* data, we considered the model with best accuracy (73.3%) belonging to a super peer. The size of this model (in terms of number of support vectors) was 3911. This simulation was comprised of 177 peers including the super peer in consideration. So, broadcasting this model to 176 peers (3911 x 176) of the network still incurred twice the relative communication cost of TRedSVM, as depicted in Figure 4.

Classification Accuracy per Hubness Level: To provide further insight into the performance of TRedSVM, we compared the average classification accuracy of all examined approaches separately for the peers of each hubness level. Figure 5 shows that for super-peers (Level-1), all methods attain comparable accuracy. This is natural to expect, since super-peers having adequately large training sets and thus gain nothing from model exchange. This explains also the good

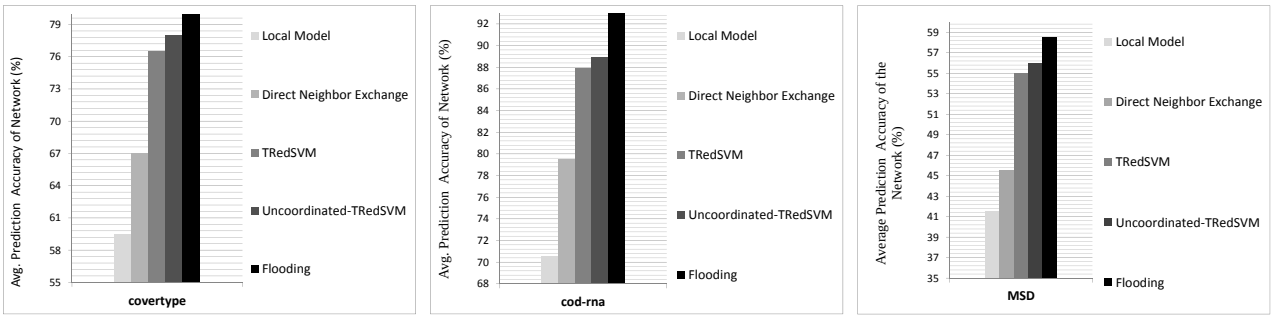


Figure 3: Average prediction accuracy of TRedSVM as compared with *Local Models*, *Direct Exchange Method* and *Flooding*

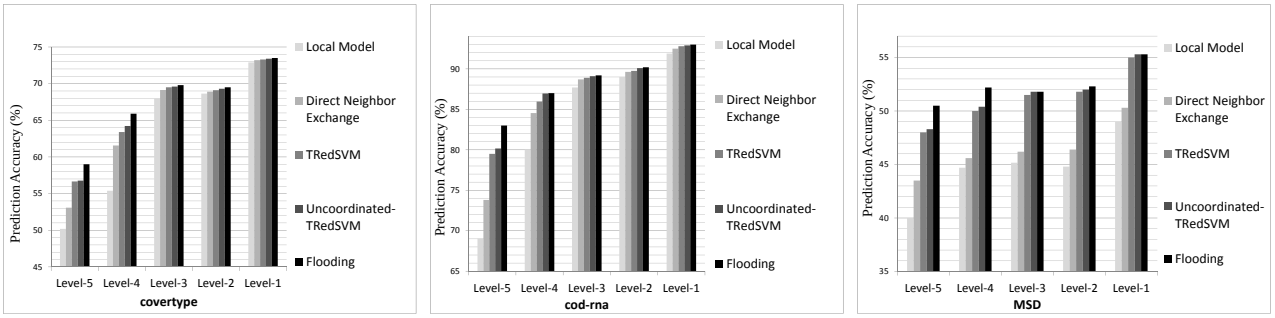


Figure 5: Average classification accuracy of peers at each hubness level and for each method

performance of the simple baseline (Local Model) that involves no exchange of models. However, for peers at lower levels, and especially for week peers (Level-5), TRedSVM is able to provide important improvements in accuracy. It is worth to note that in the case of the covertype data set, at weak peers (Level-5) the simple baseline (Local Model) achieves accuracy close to a constant classifier. The method that performs exchange only between direct models attains accuracy only slightly better than random. TRedSVM however helps a vast majority of weak peers to improve their accuracy as much as possible, as shows the comparison with the upper bound of the flooding method.

Accuracy Communication Cost Trade-off: Finally, we examine the impact of parameter η at each transitive step in TRedSVM. Figure 6 depicts in a combined way prediction accuracy and communication cost with respect to the percentage of support vectors used in RSVM. As expected, communication cost increases steadily with increasing η values. Prediction accuracy also increases with increasing η values but only up to a threshold ($\eta = 10$ for covertype and cod-rna data), after which accuracy becomes steady. This clearly shows the intuition behind the principle we follow in TRedSVM, i.e., the need to keep the size of exchanged models under control, because otherwise communication cost increases rapidly without any gain in accuracy.

Due to space limitation, we could not include our experimental results for data replication and data and model centralization schemes, which also exhibit the effectiveness of the proposed method. These results will be provided in the extended version of paper or as appendix if required.

5 Discussion and Conclusion

The first important conclusion from the presented experimental results is that for the P2P systems whose topology follows a skewed distribution of peer degrees and data, the transitive propagation of TRedSVM helps the vast majority of weak peers to substantially improve their classification accuracy. Without any exchange of models between peers (simple baseline called Local Model) or with a restricted exchange (Direct Neighbors Exchange), weak peers have no good chances to receive informative models from stronger peers that have large number of informative instances. In model propagation approaches of distributed classification in P2P networks, trade-off between prediction accuracy and communication cost is crucial and a challenging reality. Results in Figure 3 and Figure 4 illustrate lower and upper bounds on accuracy and communication cost, respectively. *Direct Neighbors Exchange* exhibits a lower bound on communication cost, while *Flooding* resulted in an upper bound on prediction accuracy. TRedSVM fitted quite well in these two bounds by incurring slightly larger

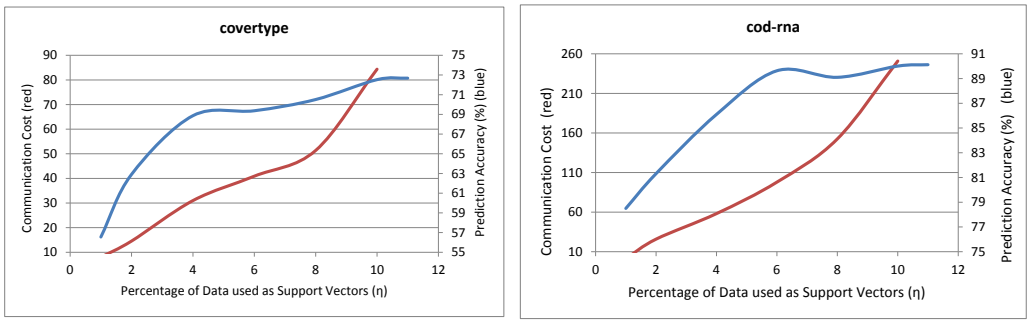


Figure 6: How prediction accuracy and communication cost varies with respect to η (Colored)

communication cost than the lower bound, and smaller by large magnitudes than in-feasible naive flooding. Moreover, controlling the model sizes with respect to η_p at transitive steps is a useful idea, since after a certain threshold of model size, accuracy is not improved further and communication cost is paid without a reason, as illustrated in Figure 6. Therefore, the optimization of the trade-off between accuracy and communication cost makes TRedSVM scalable in large P2P networks.

References

[1] I. M. J. B. Alberto Medina, Anukool Lakhina, *Brite: Boston university representative internet topology generator*, 2002.

[2] N. Andrade, E. Santos-Neto, F. Brasileiro, and M. Ripeanu, *Resource demand and supply in bittorrent content-sharing communities*, Computer Networks, 53 (2009), pp. 515–527.

[3] H. H. Ang, V. Gopalkrishnan, S. C. Hoi, and W. K. Ng, *Classification in p2p networks with cascade support vector machines*, ACM Transactions on Knowledge Discovery from Data (TKDD), 7 (2013), p. 20.

[4] A.-L. Barabsi and R. Albert, *Emergence of scaling in random networks*, Science, 286 (1999), pp. 509–512.

[5] T. Bertin-Mahieux, D. P. Ellis, B. Whitman, and P. Lamere, *The million song dataset*, in Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011), 2011.

[6] K. Bhaduri, R. Wolff, C. Giannella, and H. Kargupta, *Distributed decision-tree induction in peer-to-peer systems*, Stat. Anal. Data Min., 1 (2008), pp. 85–103.

[7] C.-C. Chang and Lin, *Libsvm: A library for support vector machines*, ACM Transactions on Intelligent Systems and Technology, 2 (2011), pp. 27:1–27:27.

[8] S. Datta, C. Giannella, and H. Kargupta, *Approximate distributed k-means clustering over a peer-to-peer network*, IEEE Trans. on Knowl. and Data Eng., 21 (2009), pp. 1372–1388.

[9] M. Karakaya, I. Körpeoğlu, and O. Ulusoy, *Counteracting free riding in peer-to-peer networks*, Comput. Netw., 52 (2008), pp. 675–694.

[10] D. Kempe, A. Dobra, and J. Gehrke, *Gossip-based computation of aggregate information*, in Proceedings of the 44th IEEE FOCS, 2003.

[11] N. Koenigstein, Y. Shavitt, E. Weinsberg, and U. Weinsberg, *On the applicability of peer-to-peer data in music information retrieval research*, in Proceedings of the 11th ISMIR 2010, pp. 273–278.

[12] D. Korzun and A. Gurtov, *Survey on hierarchical routing schemes in flat distributed hash tables*, Peer-to-Peer Networking and Applications, 4 (2011), pp. 346–375.

[13] D. Korzun and A. Gurtov, *Hierarchical architectures in structured peer-to-peer overlay networks*, Peer-to-Peer Networking and Applications, (2013), pp. 1–37.

[14] W. Kowalczyk, M. Jelasity, and A. E. Eiben, *Towards data mining in large and fully distributed peer-to-peer overlay networks*, in In Proceedings of BNAIC03, 2003, pp. 203–210.

[15] Y.-J. Lee and O. L. Mangasarian, *Rsvm: Reduced support vector machines.*, in SDM, vol. 1, SIAM, 2001, pp. 325–361.

[16] P. Luo, H. Xiong, K. Lü, and Z. Shi, *Distributed classification in peer-to-peer networks*, in Proceedings of the 13th ACM SIGKDD, ACM, 2007, pp. 968–976.

[17] A. Malatras, *State-of-the-art survey on p2p overlay networks in pervasive computing environments*, Journal of Network and Computer Applications, (2015).

[18] O. Papapetrou, W. Siberski, and S. Siersdorfer, *Efficient model sharing for scalable collaborative classification*, Peer-to-Peer Networking and Applications, 8 (2015), pp. 384–398.

[19] S. Saroiu, K. P. Gummadi, and S. D. Gribble, *Measuring and analyzing the characteristics of napster and gnutella hosts*, Multimedia Syst., 9 (2003), pp. 170–184.

[20] D. Stutzbach, S. Zhao, and R. Rejaie, *Characterizing files in the modern gnutella network*, Multimedia Systems, 13 (2007), pp. 35–50.

[21] R. Wolff, K. Bhaduri, and H. Kargupta, *Local l2 thresholding based data mining in peer-to-peer systems*, in SIAM SDM, 2006, pp. 430–441.

Towards Scalable Graph Analytics on Time Dependent Graphs

Suraj Poudel* Roger Pearce† Maya Gokhale†

Abstract

In many graph applications, the structure of a graph changes dramatically as a function of time. Such applications require temporal analytics to fully characterize the underlying data. Significant research into scalable analytics of static graphs (where the structure remains fixed) on High Performance Computing (HPC) systems has enabled new capabilities, but solutions for scalable analysis of temporal graphs are lacking. We present our ongoing work to annotate a scale-free graph in distributed memory with temporal metadata to model a time dependent graph. We demonstrate time dependent graph analytics, and provide initial scalability results on a data-intensive HPC cluster.

Our approach allows an edge to have local access to its temporal metadata and thus facilitates algorithms to make various temporal decisions, e.g., either to accept or discard the current edge at current time. We performed experiments with large scale internet traffic flows extracted from CAIDA datasets on a data-intensive HPC Cluster (Catalyst at LLNL). Using this dataset, graph representation, and the HavoqGT [1] framework, we successfully implemented and evaluated time dependent Connected Vertices, Single Source Shortest Path (TD-SSSP) and Betweenness Centrality. Finally, Betweenness Centrality was used to test the scalability of framework to 128 compute nodes of Catalyst.

Keywords: Time Dependent Graph, Graph Analytics, Distributed Graph, HPC.

1 Introduction

Time dependent graphs, also referred to as temporal graphs or time varying graphs, include temporal information in addition to the topological structure. Edges in time dependent graphs may not continuously exist and are a function of time, unlike static graphs where only a single topological structure exists. Systems with dynamic topological structure are commonly represented as aggregated graphs, e.g., as weighted graphs where weights are assigned as number of times edges occur, or the percentage of total time the edge exists. The aggregated representation is used, in part, because many tools are available for analyzing static graphs, while there is an absence of solutions for large temporal graphs. A recent review of temporal networks [2] discusses the importance and methods to analyze the temporal structure and model for understanding its impact on the behavior of the dynamical systems.

Analytics on time dependent graphs has been applied in many fields. Some examples include management and planning of complex networks [3, 4], designing strategies for containing the spread of malware in mobile devices [5], analysis of the circadian patterns of Wikipedia editorial activity to estimate the geographical distribution of editors [6]. With temporal information, existing graph analytics can be processed with respect to some global time parameters such as *Start time*, the time at which a traversal begins from a vertex and/or *Waiting time*, the upper bound on the time a traversal can halt in a vertex before continuing its traversal to its neighbors as shown in [7]. As an example, time dependent single source shortest path (TD-SSSP) in a road network could yield optimal start time for a fuel efficient travel from a source to destination including amount of time to wait in a city before proceeding to the next. Also, the temporal information can be utilized to benefit towards other purposes, e.g., decaying weights, establishing probabilities with time. In this work we experimented with similar analytics with respect to these time parameters in large scale-free distributed graphs as an illustration of the capability of the framework we have developed.

Significant research in the HPC community has focused on the analysis of large static graphs [8, 9, 10], fueled in large part by the introduction of the Graph500 benchmark [11]. Much of this work has focused on processing the largest graph topology possible, on some of the world's largest supercomputers. However, little attention has been given to the processing of graph metadata (annotations on vertices and edges) or the temporal nature of many graph datasets.

This work extends HavoqGT [1] to add temporal metadata to the edges in the existing partitioned-graph topology and analyze the resulting metadata-annotated graph with various time dependent algorithms. We aim to demonstrate a new set of capabilities for analyzing

*University of Alabama at Birmingham, spoudel@uab.edu
†Lawrence Livermore National Lab, {rpearce,maya}@llnl.gov

large scale temporal graphs on modern HPC clusters, by performing scaling studies with existing time dependent analytics. Our main contributions in this work are:

1. Extending the capability in HavoqGT to support implementation of time dependent graph algorithm;

2. Demonstrate temporal graph analytics across two example time parameters – traversal start and wait time;

3. Show initial scalability results of the extended HavoqGT for time dependent algorithms on a HPC cluster.

2 Related Work

Algorithmic optimizations to compute time-dependent algorithms such as shortest paths over large graphs by Ding et al. [7] suggest the need for faster computation for temporal graph analysis. A use case study by Cattuto et al. [18] shows the model of a time-varying social network as a property graph in the Neo4j graph database that focuses on database-type queries based on time, rather than algorithms for graph analytics such as time dependent SSSP. A distributed programming approach by Simmhan et al. [19] uses temporally iterative BSP model on more specific time-series graphs where edge parameters change in time more frequently (than topology). Han et al. [20] proposes time-locality aware in-memory layouts for snapshots of temporal graphs for optimal performance on batch-scheduling of graph computations. However, the paper states that benefits of suggested optimizations are not prominent in more network-constrained environments. In our work, we use our edge based-partitioning with distributed delegates to partition edges (duplicating for each time duration it exists) and use asynchronous visitor model to design graph analytics to execute on distributed-memory HPC systems.

Related research into efficient multithreaded betweenness centrality by Madduri et al. [12], and distributed-memory breadth-first search [8, 9, 10] have focused on algorithm specific implementations and optimizations for static graphs on HPC Systems. Additionally, recent frameworks like PowerGraph [14], GraphLab [15], Pregel [16], GraphX [17] illustrate their use only on large static graphs. In this work, we extend an existing framework, HavoqGT, designed for graph analytics on HPC systems, for use on large scale time-depending graphs.

Table 1: **Visitor Procedures and State**

pre_visit	Performs a preliminary evaluation of the state and returns true if the visitation should proceed.
visit	Main visitor procedure
operator >	Greater than comparison used to locally prioritize the visitor in a max heap priority queue.
vertex	Stored state representing the vertex to be visited.

3 Background

3.1 HavoqGT. HavoqGT (Highly Asynchronous Visitor Queue Graph Toolkit) [1] is a framework developed based on our previous work [21, 22, 23]. It presents a new graph partitioning technique and computation model that distributes the storage, computation and communication for high-degree vertices (hubs) in large scale-free graphs by creating vertex delegates, hub vertices replicated at each partition, co-located with the edges' target to achieve a partitioned graph that reduces communication with the neighbors of hub-vertices and allows faster parallel traversal. It also provides the distributed framework that allows implementation of algorithms using an asynchronous visitor abstraction to be run in parallel and distributed environments on large scale-free graphs.

This work uses the distributed asynchronous visitor queue abstraction from HavoqGT to provide the parallelism and to create a data-driven flow of computation. Traversal algorithms are created using a visitor abstraction, which allows an algorithm designer to define vertex-centric procedures to execute on traversed vertices with the ability to pass visitor state to other vertices [23]. This abstraction requires certain visitor procedures and state to be defined in the visitor as shown in Table 1. Algorithm implementations presented in this paper follow this abstraction. When an algorithm needs to traverse to another vertex, it dynamically creates a new visitor and pushes it into the visitor queue. When an algorithm begins, an initial set of visitors are pushed on the queue and the framework's driver invokes the traversal which runs the asynchronous traversal to completion.

Our previous work [23] showed that HavoqGT provided excellent scalability on large scale-free static graphs up to 131K cores of the IBM BG/P, and outperformed the best known Graph500 performance on BG/P Intrepid. In this work, we build upon this existing scalable framework to provide graph analytics for time dependent graphs.

3.2 Datasets. To experiment with the time dependent graphs, we required a dataset representative of time varying large scale-free graphs. The CAIDA [24] dataset consist of ∼8 TB of publicly available large anonymized passive traffic traces and packet header traces saved per direction per server collected over 39 1-hour periods. As packets represent finer-granularity in network communication mostly governed by the underlying protocol, higher abstraction of network communication i.e. the connection is more likely to represent and resemble numerous other real-world graphs.

Flows between two communicating sockets were derived by binning the packets exchanged between sockets by their start time into fixed-sized time windows. All packets communicated between two sockets in a defined time window form a flow. As shown in Figure 1, Packet 1 and Packet 2 are combined to form a flow as both of these packets fall within the 0.0001s time window. A fixed timeout value from the start time of the first packet was used to gather a flow's packets (described below):

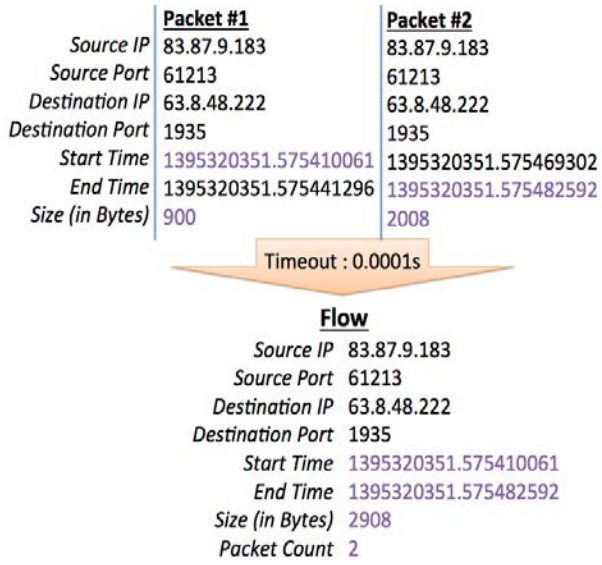


Figure 1: Figure showing how two packets are merged to form the flow. Flow begins with the first packet and then collects all the subsequent packets until the timeout. Notice that the parameters size and packet count are accumulated over all the packets in the flow. Also the start and the end time of the flow merge the time intervals of each packets into one interval.

DEFINITION 3.1. *A flow from X to Y, between two communicating socket's X and Y, corresponds to all the packets p_i transmitted from X to Y, where $i = m$ to n $(m \leq n)$ and p_i is ordered in non-decreasing start_time, such that*
1. $start\_time\,(p_m)$ is the flow's start time
2. $end\_time\,(p_n) - start\_time\,(p_m) \leq timeout$, and
3. $start\_time\,(p_{n+1}) > (start\_time\,(p_m) + timeout)$

Applying this algorithm with timeout of 0.0001s, we generated ∼141 billion flows with the ∼8 TB CAIDA datasets. The generated flow files were split into smaller chunks, and stored on a Lustre distributed filesystem.

3.3 Time Dependent Single Source Shortest Path. A SSSP in a weighted graph (with time delay edge weight) is a query requesting the shortest time path from a source vertex to all the other vertices. In a time dependent graph, the SSSP is influenced largely by the current timestamp of the graph traversal because edges (unlike in time independent graphs) exists as a function of time.

In TD-SSSP, the arrival time at destination vertices depends on when the traversal begins at the source vertex, referred to as the traversal *start time*. Traversal may arrive to a vertex at different times because edges that exist at one point in time of traversal can be different (might not exist or have different edge weights) at another point in time of traversal at the same parent in current path. The notion of the traversal *waiting time* on a vertex further adds to the dynamic behavior of the traversal by having a potential to provide more (sometimes optimal) choices of path with the wait. As an example, the edge delay of a flow, the time required for the flow to reach the destination IP, at time t2 in addition to the waiting time in the current IP can be less than the edge delay at any other time t1 (and t1 < t2). So, TD-SSSP answers queries of finding the optimal start time and wait time that minimizes the total traversal cost.

3.4 Temporal Betweenness Centrality. Betweenness Centrality is the measure of the extent to which a vertex in a graph is important or central, with respect to its presence on the shortest path between pairs of other vertices [25]. The betweenness centrality of a vertex is the ratio of the number of shortest paths between pairs of other vertices passing through that vertex to the total number of shortest paths between pairs in the graph. Temporal Betweenness Centrality measures the importance of the vertex in the temporal shortest paths between pairs of other vertices i.e. the temporal betweenness for a vertex on a given time interval is the sum of the proportion of all the temporal shortest paths through that vertex to the total number of temporal shortest paths over all pairs of vertices for each time interval in the given time interval. This algorithm relies on calculation of the temporal shortest path across

pairs at different time interval, as described in Section 3.3.

4 Distributed Time Dependent Graph Construction

To process a time dependent graph in HavoqGT, the graph must be ingested from a raw unpartitioned input source. The input is an unordered edge list with edge metadata and temporal information. In two passes, we first partition the graph topology and second, decorate the graph with the corresponding metadata.

The graph topology is partitioned using the distributed delegate techniques described in [23]. The edge list of the input graph is provided to the framework using the source and destination IP pair from the metadata list. Several communication phases occur during the partitioning phase to partition and balance the topology. We avoid bundling the edge with its metadata during the partitioning process, which could unnecessarily increase the communication volume. Instead, in the second pass edge metadata is directly sent to the edge's final location using the asynchronous visitor abstraction.

Using the asynchronous visitor model, a visitor is created for each edge metadata and pushed into the global visitor queue, which pushes the visitor to the partition owning the source vertex of the edge, thus relaying the metadata to the outgoing edge of the corresponding vertex. However, for the high out-degree vertices which have been *delegated*, not all edges can be located on a single partition. In such case, the controller broadcasts the edge metadata to hub delegates and registers it to outgoing edge of the delegate in the partition that owns the edge for that metadata.

In our experiments, we successfully ingested 1.35 billion edges (one hour CAIDA flows) on the Catalyst cluster in around 5 minutes using 32-compute nodes and 24 processes per node. The raw edge input typically resides on a shared resource such as a distributed Lustre file system, sometimes causing large performance variations depending on current load.

5 Distributed Graph Analytics

In this section, we show how we use multiple, temporal-metadata entries per edge to represent a time dependent graph, and show implementation details of time dependent single source shortest path (TD-SSSP) and betweenness centrality (TD-BC).

5.1 Time Dependent Graphs. Each flow presents packets sent from one socket to the other for a time interval [a start time, an end time). Figure 1 shows a typical flow with its start and end time along with other

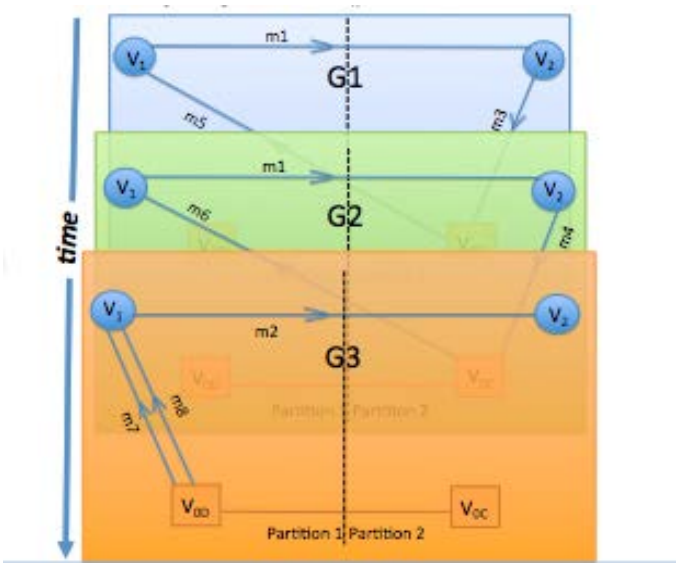


Figure 2: An example time dependent graph over three time intervals

parameters describing it(e.g., total size transferred in the flow is 2908). Two communicating sockets can have different flows between them at different time intervals. We use flows to combinely represent edges and their metadata. The communication from a socket to the other (link) in the flow represents the directed edge and all the other parameter including time interval in the flow represents the metadata for that edge.

Let us denote the i-th flow passes from socket u_i to v_i and its start time and end time are ${t_s}^i$ and ${t_f}^i$ respectively and define $[0, T]$ as the global time interval of the dataset i.e. there is no flow with end time beyond time T. Then, $e_i = (u_i, v_i)$, the i-th edge, represents link in i-th flow and $m_i = \left(s_i = {t_s}^i, f_i = {t_f}^i, other_i\right)$, the metadata for the i-th edge contains all the other information for the i-th flow, where u_i is the source vertex, v_i is the destination vertex , s_i is the start time of edge i, f_i is the finish time of edge i and $other_i$ represents other parameters involved in the flow. Let (T_s, T_f) be any time interval window. If $M\,(T_s,\ T_f)$ is the set of metadata such that $T_s \leq start\_time(m_i \in M\,(T_s,\ T_f)) \leq T_f$, then the edge corresponding to each metadata m_i forms a set of edges $E\,(T_s,\ T_f)$ that exists in this time interval $(T_s,\ T_f)$. Thus, the graph $G\langle T_s,\ T_f\rangle\,(V, E\,(T_s,\ T_f))$ represents the time dependent graph in time interval $(T_s,\ T_f)$. So, different sets of graph topology with different edge parameters at each edges can be seen for $0 \leq T_s < T_f \leq T$.

Figure 2 shows an example of a time dependent graph modeled from multi-metadata per edge. If M(0,

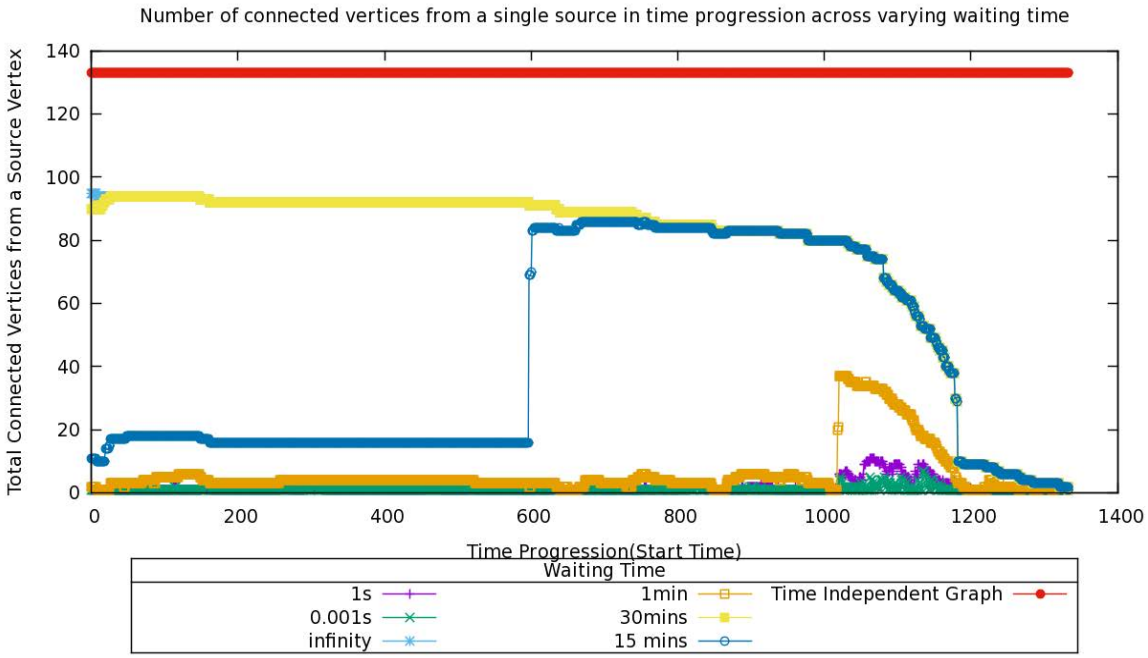


Figure 3: Plot of number of connected vertices from a single source with respect to the change in start time experimented over different waiting time. Time independent traversal from this source has 133 connected vertices while the time dependent traversal creates significant variation that is further affected by the waiting time spent on each vertex.

t1) = $\{m1,\ m3,\ \ m5\}$, M(t1, t2) = $\{m1,\ m4,\ m6\ \}$ and M(t2, t3) = $\{m2,\ \ m7,\ m8\}$ are three sets of metadata over time series 0, t1, t2 and t3, then three graphs are obtained at those time-intervals as shown in Figure 2 as G1, G2 and G3. V_{0D} and V_{0C} form the distributed partitioned hub vertex generated by the graph construction method discussed in [23] , with V_{0C} the controller and V_{0D} the delegate.

5.2 Algorithms Implementation Details. The addition of temporal metadata on the graph topology supports implementation of various time dependent algorithm in HavoqGT. We discuss the implication of the temporal information in the graph with respect to the connectedness of a single vertex and show two algorithm implementations, time dependent single source shortest path algorithm and temporal betweenness centrality. Our framework simplifies the design of algorithms for time dependent algorithms to be run on HPC systems by providing a programming abstraction. Algorithms shown below highlights the use cases of such abstraction along with the local access of the large temporal information to achieve scalable graph analytics on large time dependent graphs.

5.2.1 Connected Vertices. A time dependent version of connected components is used to count connected vertices from a single source for different start times and waiting times. *Waiting time* is the maximum time a traversal can wait at each vertex before visiting its neighbors; it can vary the range of traversable edges that has significance in planning route in complex networks. The plot in Figure 3 shows the connected vertices from a single source in the CAIDA data. The count of connected vertices remains constant in the time-flattened version of our graph (traditional static graph model). In the time dependent graph, the start time and the waiting time has significant impact on the count. Beginning the traversal at a later time showed comparatively greater connectivity for waiting time 1-15 mins, which in context of TD-SSSP could yield in a shorter arrival time at the destination due to wider range of path choices.

5.2.2 Time Dependent Single Source Shortest Path. We use a label correcting approach to compute the time dependent single source shortest path (TD-SSSP) which records the shortest arrival time at each vertex, and traverses the neighbors that exist between *arrival time* and *arrival time* + *waiting time*.

The TD-SSSP begins by visiting the outgoing edges of a source vertex that exist within the start-time and waiting-time. The timestamp of traversed edges determines the new arrival time at neighboring vertices. The TD-SSSP visitor is shown in Algorithm 1. When visiting a vertex at a specific arrival time, it uses the following three conditions: (a) If the arrival time at the current vertex is less than the last recorded smallest arrival time, record this time as the shortest arrival time (Alg. 1, line 5); (b) Only visit the vertex if the new arrival time is registered (Alg. 1, line 8); (c) Only traverse the neighbors that exist for waiting time period after the arrival time at this vertex (Alg. 1, line 15).

Algorithm 1 Time Dependent Single Source Shortest Path

```
1:  visitor_state : vertex ← vertex to be visited
2:  visitor_state : arrival_time ← path arrival time
3:  visitor_state : parent ← path parent

4:  procedure PRE_VISIT(vertex_data)
5:      if arrival_time < vertex_data.arrival_time then
6:          vertex_data.arrival_time ← arrival_time
7:          vertex_data.parent ← parent
8:          return true
9:      end if
10:     return false
11: end procedure

12: procedure VISIT(graph, visitor_queue)
13:     for all e_i ∈ out_edges(graph, vertex) do
14:         metadata ← edges_metadata(e_i)
15:         not_earlier ← arrival_time + waiting_time
16:                 ≥ metadata.start_time
17:         not_later ← arrival_time < metadata.end_time
18:         edge_exists ← not_earlier and not_later
19:         if edge_exists == true then
20:             new_arrival_time ← metadata.end_time
21:             new_visitor ← td_sssp_visitor ( e_i.target
22:                     , new_arrival_time, vertex)
23:             visitor_queue.push(new_visitor)
24:         end if
25:     end for
26: end procedure

27: procedure OPERATOR>( td_sssp_visitor a,
    td_sssp_visitor b )
28:     return a.arrival_time > b.arrival_time
29: end procedure
```

We also record the parent of each vertex in the shortest path found from the source to this vertex, thus forming a shortest path tree from the source. This tree yields the earliest arrival time from the source to each vertex and also the actual waiting time spent at each vertex in the shortest path from the source.

5.2.3 Temporal Betweenness Centrality. We identified two major steps in evaluating the unnormalized but exact betweenness centrality: a) Computing temporal shortest path between all pairs, and b) Accumulating the count per vertex of the number of times the vertex is traversed in a path from the leaves of the shortest path tree of a source to its root per source. We extended the TD-SSSP (5.2.2) to start the traversal from all the vertices and kept the record of the temporal shortest path from each source. Next, we employed a distributed recursive algorithm in HavoqGT to process the TD-SSSP data per source to obtain the count of shortest paths through each vertex.

The recursive algorithm starts with computing the out-degree of each vertex per shortest path tree from a single source. This out-degree count per shortest path tree will help us identify whether or not all the children of a vertex have been processed so that it can decide if the path count collected so far is the complete count to pass it up to its parent. For each source, we start traversal from all vertices in source's shortest paths that have no children. The vertex's parent is then traversed and its child count decremented by one (Alg. 2, line 5) which signifies of a completed traversal at that child. The count passed from that child represents total number of shortest paths that go through that child and thus is added to the total count of shortest path through this node (Alg. 2, line 6). The parent will check to see if its child count is zero (Alg. 2, line 7). If not, it terminates that traversal and returns (Alg. 2, line 10). Eventually, one of its children will come back to this parent and set the child count to zero at which point, the parent starts its traversal towards its parent (Alg. 2, line 8).

6 Experimental Results

All our experiments have been executed on 150 TeraFLOP/s Catalyst, Linux HPC Cluster located on Lawrence Livermore National Laboratory. The results are based on application of experimental distributed graph analytics on one-hour period CAIDA flows having 1.35 billion edges (∼190GB).

Figure 3 summarizes the results obtained from traversing the time dependent CAIDA graph from an example single source to record its connectivity to other vertices at various start time and waiting time. The variation in connectivity at various start time (plotted as x-axis in Figure 3) is further analyzed for different waiting time. Globally assigned waiting time for a

Algorithm 2 Temporal Betweenness Centrality

```
1:  visitor_state : vertex ← vertex to be visited
2:  visitor_state : source ← source of the traversal
3:  visitor_state : count ← shortest path count

4:  procedure PRE_VISIT(vertex_data)
5:      vertex_data[source].child_count ←
            vertex_data[source].child_count − 1
6:      vertex_data[source].shortest_path_count ←
            vertex_data[source].shortest_path_count
                +count
7:      if vertex_data[source].child_count == 0 then
8:          return true
9:      else
10:         return false
11:     end if
12: end procedure

13: procedure VISIT(graph, visitor_queue)
14:     parent ← SSSP[source].parent_of(vertex)
15:     source_data ← vertex_data[source]
16:     count ← source_data.shortest_path_count
17:     new_visitor ← bc_visitor(parent, source, count)
18:     visitor_queue.push(new_visitor)
19: end procedure
```

traversal shows significant effect in changing the results of various well-known graph algorithms and thus impacts the computation time correspondingly. The variation shown in Figure 3 shows the reality of the underlying data more accurately than the time aggregated version which would always treat the connectivity the same across all time ranges.

6.1 Temporal Betweenness Centrality. The unnormalized temporal betweenness centrality value of four example vertices with respect to the path waiting time is shown in Figure 4. With the experimented CAIDA dataset, the plotted vertices show an increasing trend in BC value with increasing waiting time at vertex, as greater waiting time can make more neighbors traversable. The implication of such results can be to find vertices with maximum increasing trend of becoming central within a desired time range.

The computation time of temporal betweenness centrality is shown in Figure 5. The performance varies greatly as a function of traversal start-time and waiting-time. The variance for a range of start times is shown using 64-compute nodes of the Catalyst cluster. For each waiting-time value (x-axis), 20 different starting-times are shown by the variance bars. As the waiting-time is increased, the traversal complexity increases as

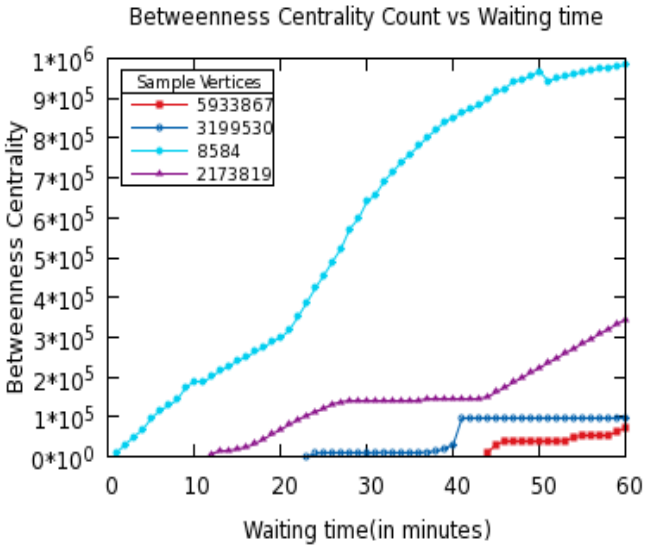


Figure 4: Plot of unnormalized exact Betweenness Centrality of four example vertices for varying waiting time of the shortest path traversal for a fixed starting time of traversal.

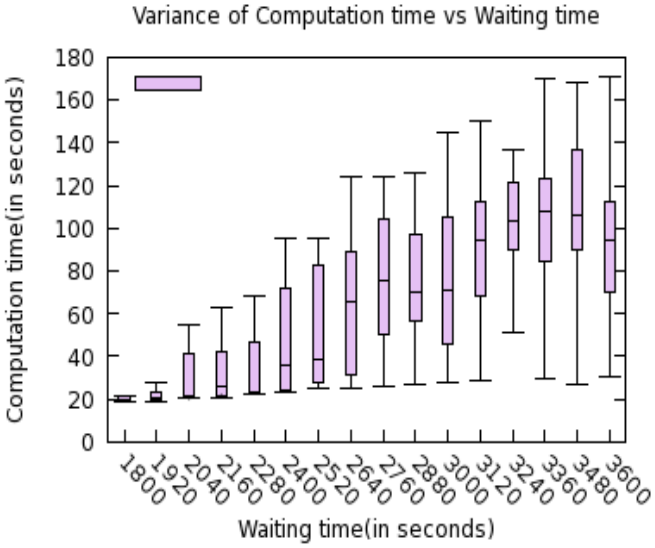


Figure 5: Plot of variance of the computation time for exact betweenness centrality executed on 64 compute nodes for traversal start time within first 20 minutes of the test data with respect to waiting time period from half-an-hour to the full hour. Intuitively, more computation is required for higher waiting time at earlier start times as more neighboring edges are available as we begin early and wait longer.

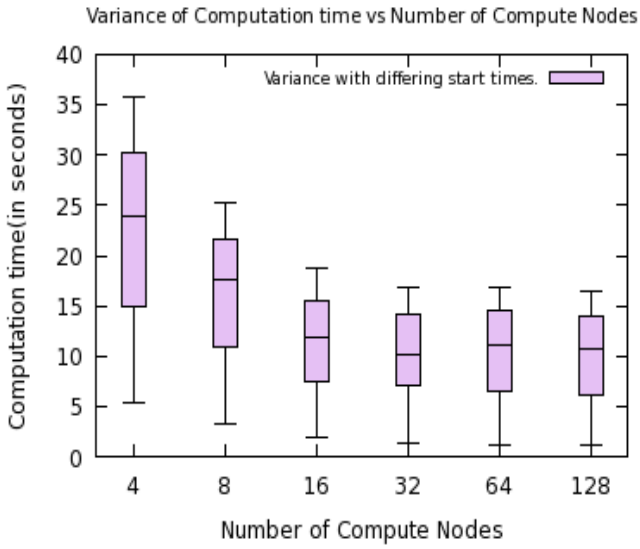


Figure 6: Plot of variance of computation time across various start time with respect to the number of compute nodes used to process the time dependent exact betweenness centrality.

the number of eligible paths increases.

Finally, Figure 6 shows the result of testing strong scaling of the betweenness centrality executed on the Catalyst cluster. Variation in computation time with respect to differing start times was computed for an infinite waiting time in order to test the scalability of this framework. The computation time decreases with an increasing number of compute nodes, and scales near-linearly up to approximately 32 compute nodes. This strong-scaling limit is not uncommon for distributed graph analytics. In our future work, we will investigate weak-scaling to larger datasets.

7 Summary

We presented our ongoing work to annotate a scale-free graph in distributed memory with temporal metadata to model a time dependent graph. With the increased capability of our distributed framework for high performance temporal graph analytics, HavoqGT, we have shown implementations for multiple important graph analytics on time dependent graphs. We provide initial scalability results on a data-intensive HPC cluster for temporal algorithms using two global time parameters *start-time* and *waiting-time*, to provide analysis on time dependent graphs derived from CAIDA datasets.

8 Acknowledgments

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 (LLNL-CONF-676816). Funding was partially provided by LDRD 13-ERD-025. Experiments were performed at the Livermore Computing facility.

References

[1] "HavoqGT," in *http://havoqgt.bitbucket.org/*.

[2] P. Holme and J. Saramaki, "Temporal networks," *Physics Reports*, vol. 519, no. 3, pp. 97 – 125, 2012. Temporal Networks.

[3] A. Orda and R. Rom, "Shortest-path and minimum-delay algorithms in networks with time-dependent edge-length," *Journal of the ACM (JACM)*, vol. 37, no. 3, pp. 607–625, 1990.

[4] A. K. Ziliaskopoulos and H. S. Mahmassani, "Time-dependent, shortest-path algorithm for real-time intelligent vehicle highway system applications," *Transportation research record*, pp. 94–94, 1993.

[5] J. Tang, C. Mascolo, M. Musolesi, and V. Latora, "Exploiting temporal complex network metrics in mobile malware containment," in *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2011 IEEE International Symposium on a*, pp. 1–9, IEEE, 2011.

[6] T. Yasseri, R. Sumi, and J. Kertész, "Circadian patterns of wikipedia editorial activity: A demographic analysis," *PloS one*, vol. 7, no. 1, p. e30091, 2012.

[7] B. Ding, J. X. Yu, and L. Qin, "Finding time-dependent shortest paths over large graphs," in *Proceedings of the 11th international conference on Extending database technology: Advances in database technology*, pp. 205–216, ACM, 2008.

[8] A. Yoo, A. Baker, R. Pearce, and V. Henson, "A scalable eigensolver for large scale-free graphs using 2D graph partitioning," in *Supercomputing*, pp. 1–11, 2011.

[9] A. Buluç and K. Madduri, "Parallel breadth-first search on distributed memory systems," in *Supercomputing*, 2011.

[10] F. Checconi, F. Petrini, J. Willcock, A. Lumsdaine, A. R. Choudhury, and Y. Sabharwal, "Breaking the speed and scalability barriers for graph exploration on distributed-memory machines," in *Supercomputing*, 2012.

[11] "The Graph500 benchmark," in *www.graph500.org*.

[12] K. Madduri, D. Ediger, K. Jiang, D. Bader, and D. Chavarria-Miranda, "A faster parallel algorithm and efficient multithreaded implementations for evaluating betweenness centrality on massive datasets," in *Parallel Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, pp. 1–8, May 2009.

[13] A. Buluç and K. Madduri, "Parallel breadth-first search on distributed memory systems," in *Proceed-*

ings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, SC '11, (New York, NY, USA), pp. 65:1–65:12, ACM, 2011.

[14] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin, "Powergraph: Distributed graph-parallel computation on natural graphs," in *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation*, OSDI'12, (Berkeley, CA, USA), pp. 17–30, USENIX Association, 2012.

[15] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, and J. M. Hellerstein, "Distributed graphlab: A framework for machine learning and data mining in the cloud," *Proc. VLDB Endow.*, vol. 5, pp. 716–727, Apr. 2012.

[16] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: A system for large-scale graph processing," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, SIGMOD '10, (New York, NY, USA), pp. 135–146, ACM, 2010.

[17] R. S. Xin, J. E. Gonzalez, M. J. Franklin, and I. Stoica, "Graphx: A resilient distributed graph system on spark," in *First International Workshop on Graph Data Management Experiences and Systems*, GRADES '13, (New York, NY, USA), pp. 2:1–2:6, ACM, 2013.

[18] C. Cattuto, M. Quaggiotto, A. Panisson, and A. Averbuch, "Time-varying social networks in a graph database: A neo4j use case," in *First International Workshop on Graph Data Management Experiences and Systems*, GRADES '13, (New York, NY, USA), pp. 11:1–11:6, ACM, 2013.

[19] Y. Simmhan, N. Choudhury, C. Wickramaarachchi, A. Kumbhare, M. Frincu, C. Raghavendra, and V. Prasanna, "Distributed programming over time-series graphs," in *Parallel and Distributed Processing Symposium (IPDPS), 2015 IEEE International*, pp. 809–818, May 2015.

[20] W. Han, Y. Miao, K. Li, M. Wu, F. Yang, L. Zhou, V. Prabhakaran, W. Chen, and E. Chen, "Chronos: A graph engine for temporal graph analysis," in *Proceedings of the Ninth European Conference on Computer Systems*, EuroSys '14, (New York, NY, USA), pp. 1:1–1:14, ACM, 2014.

[21] R. Pearce, M. Gokhale, and N. M. Amato, "Multithreaded asynchronous graph traversal for in-memory and semi-external memory," in *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '10, (Washington, DC, USA), pp. 1–11, IEEE Computer Society, 2010.

[22] R. Pearce, M. Gokhale, and N. M. Amato, "Scaling techniques for massive scale-free graphs in distributed (external) memory," in *Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing*, IPDPS '13, (Washington, DC, USA), pp. 825–836, IEEE Computer Society, 2013.

[23] R. Pearce, M. Gokhale, and N. M. Amato, "Faster parallel traversal of scale free graphs at extreme scale with vertex delegates," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '14, (Piscataway, NJ, USA), pp. 549–559, IEEE Press, 2014.

[24] CAIDA, "The CAIDA UCSD Anonymized Internet Traces." `http://www.caida.org/data/overview/`, 2008.

[25] L. C. Freeman, "A set of measures of centrality based on betweenness," *Sociometry*, pp. 35–41, 1977.

GSK: **G**raph **S**parsification as a **K**napsack problem formulation

Hongyuan Zhan Kamesh Madduri
The Pennsylvania State University
hzz5039@psu.edu, madduri@cse.psu.edu

Abstract

Graph edge sparsification, or filtering a substantial number of edges in a methodical manner, has several graph analysis-related uses. In this paper, we present GSK, a new problem formulation for **g**raph edge **s**parsification based on the **k**napsack problem. We show that several prior methods for edge sparsification can be expressed as instances of the proposed GSK problem. We implement a fast linear-time approximation scheme to solve the GSK optimization problem. We also present a preliminary empirical evaluation of GSK-based sparsification strategies, comparing them to prior methods on a collection of test graphs.

Keywords: sparsification, knapsack, edge centrality.

1 Introduction

We present a simple formulation for graph sparsification based on the 0/1 knapsack problem. Informally, the 0/1 knapsack problem can be stated as follows: given a collection of items, each with an associated weight and value, the goal is to choose a subset of the items such that the cumulative value of the chosen items is maximized, and the total weight of the selected subset of items is less than or equal to a user-specified weight budget. The main idea of our GSK formulation is to treat each edge in the graph as an item and to associate weights and values to them, so that sparsification strategies can be appropriately defined.

When sparsifying a graph, we would like to preserve structural or statistical properties of the original graph as much as possible, but reduce the number of edges. We assume that the vertex set V remains the same. Let $\tilde{G}(V, \tilde{E})$ denote the graph after sparsification. We would like to minimize the loss of information, modeled by an appropriate loss function $\mathcal{L}(G, \tilde{G})$. Since the set of vertices is unchanged, we may write $\min \mathcal{L}(G, \tilde{G})$ as $\max f(\tilde{E})$ for some function f. User-defined edge weights or costs $\{c_e \mid e \in E\}$ provide fine-grained control over whether an edge should be included or not. A user-specified knapsack cost upper bound W can also be set appropriately, in the same units as c, to control the sparsity of the simplified network. When f is a linear function, i.e., associating a set of profits or values $\{p_e \mid e \in E\}$ with edges, graph sparsification can be written as a linear knapsack problem:

$$\text{(GSK)} \qquad \max_{\tilde{E} \subseteq E} \sum_{e \in \tilde{E}} p_e \;\text{ subject to }\; \sum_{e \in \tilde{E}} c_e \leq W.$$

A natural way to use this problem formulation would be to set profits to computed global or local edge centrality values. Edge centrality values attempt to quantitatively rank edges. Sparsification using the above knapsack formulation would then mean filtering low centrality edges, while satisfying user-specified linear constraints.

This paper makes the following contributions: In Section 2, we show that a number of prior edge sparsification methods can be expressed as special cases of the problem (GSK). We implement a fast solution strategy for this problem, discussed in Section 3. In Section 4, we perform a preliminary exploration of the design space enabled by our knapsack formulation, by encoding pairs of complementary edge centrality values as profits and costs. We also empirically evaluate these sparsification strategies and the performance of our solver on several test graph instances in Section 5.

2 Transforming Prior Methods to GSK

Graph edge sparsification is performed for a variety of reasons. We briefly discuss the motivation for each of the following strategies, and then describe how they can be recast as instances of our proposed knapsack-based problem.

2.1 Backbone Extraction. A large collection of relational data sets can be modeled as weighted graphs. Backbone extraction refers to the problem of determining a reduced representation of the original network with fewer edges. The backbone highlights the structure of the network and preserves key characteristics. Serrano et al. [23] propose a method to extract backbones from weighted graphs. First, edge weights are normalized locally. Let z_{ij} denote the weight of edge $\langle i, j \rangle$. The weight is changed to $w_{ij} = z_{ij} / \sum_{\{k \mid \langle i,k \rangle \in E\}} z_{ik}$.

The denominator is the sum of the weights of all edges connected to vertex i. In case of undirected graphs, each edge is considered twice, and weights are normalized according to vertices i and j separately. In the next step, a statistical test against a null model of a normalized edge weight distribution is used to determine edges that are statistically significant. The null model assumes that normalized edge weights of a degree-k vertex are proportional to random assignments of k sub-intervals from $[0, 1]$. Under this null model, the probability of observing an edge with normalized weight of at least w_{ij} is $\alpha_{ij} \stackrel{\text{def}}{=} 1 - (k-1)\int_0^{w_{ij}} (1-x)^{k-2} dx$. A local significance level α is selected, and if $\alpha_{ij} < \alpha$, the observed edge is considered to be statistically significant and retained. Other edges are filtered.

Foti et al. [9] consider a related approach using a non-parametric statistical test. The same weight normalization step is performed, but the statistical test is replaced by $\alpha_{ij} = 1 - \frac{1}{d_i} \sum_{\{k|\langle i,k\rangle \in E\}} \mathbf{1}\{w_{ik} \leq w_{ij}\}$. Here, $\mathbf{1}\{\cdot\}$ denotes the indicator function, which equals 1 if the expression in the bracket is true, and 0 otherwise. d_i denotes the degree of vertex i. α_{ij} is considered to be the probability of empirically observing an edge with normalized weight at least w_{ij} locally at vertex i. These two backbone extraction methods can be viewed as a special case of the knapsack problem:

$$\max_{\tilde{E} \subseteq E} \sum_{i \in V} \sum_{\langle i,j \rangle \in \tilde{E}} (\alpha - \alpha_{ij}).$$

The above problem can be considered a Lagrangian relaxation of problem (GSK), and with no constraints. This problem can be easily solved by selecting edges with positive profits, i.e., $\alpha - \alpha_{ij} > 0$.

2.2 Similarity Filters for Clustering. Satuluri et al. [22] use edge sparsification as a preprocessing step to enhance scalability of graph vertex clustering algorithms. The goal is not to sacrifice the quality of clustering results with the sparsified graph. Each edge $\langle i, j \rangle$ is evaluated based on the similarity of vertices i and j. Similarity is defined using the Jaccard score:

$$\texttt{Jac}_{ij} = \frac{|N(i) \cap N(j)|}{|N(i) \cup N(j)|},$$

where $N(i)$ refers to the set of neighboring vertices of i. The authors propose two sparsification schemes based on the Jaccard score. The global similarity method first ranks each edge in the network using Jac; then the top $s\%$ edges ranked by Jac are retained in the sparsified network. Therefore, the global similarity method could be viewed as problem (GSK) with unit edge weights, i.e., a edge cardinality constraint:

$$\max_{\tilde{E} \subseteq E} \sum_{e \in \tilde{E}} \texttt{Jac}_e \;\text{ subject to } |\tilde{E}| \leq W.$$

In addition to filtering edges globally, the authors also consider ranking edges locally at each vertex similar to [23] and [9]. Starting with a network of the same vertex set and an empty edge set, for each vertex $i \in V$, the top $\lceil d_i^\alpha \rceil$ (exponent $\alpha \in (0, 1)$) of the incident edges in the original network ranked by similarity score $\texttt{Jac}_{ij}$ are added back. The intuition is that similar vertices are more likely to reside in the same cluster, and so intra-cluster edges are retained. This problem can be viewed as an unconstrained knapsack problem by setting the profit values of edges that are to be filtered to zero, and the profit values for the edges to be retained to their Jaccard scores. We could impose additional filtering constraints to complement this local similarity filter, as we will discuss in Section 4.

2.3 Local Degree Filter. Lindner et al. [19] recently evaluate a degree-based sparsification strategy. Similar to the Local Similarity filter method, each edge is first rated using a score function, and then a filtering threshold is applied locally at each vertex. For each edge $\langle i, j \rangle$ incident on vertex i, the score function $\texttt{Deg}_{ij}$ chosen here is the degree of vertex j. For each vertex i, the top-ranked $\lceil d_i^\alpha \rceil$ of attached edges are retained. The motivation behind using vertex degree for filtering is to preserve key graph vertices, or hub vertices, after sparsification. Analogous to the Local Similarity method, this method can be written as a special unconstrained case of problem (GSK).

2.4 Other Related Sparsification Methods. Graph sparsification has been studied in several different contexts. Cut sparsifiers [3, 10] aim at creating a graph with the same set of vertices, but with fewer edges, such that every cut on the graph is preserved up to a multiplicative factor. Spectral sparsifiers by Spielman and Teng [28] is another well-known line of work. The goal here is to preserve the spectral properties of the graph Laplacian. Bonchi et al. [5] define the problem of activity-preserving graph simplification. Their approach requires additional information on activity traces in the network. The goal is to preserve observed activity traces while simplifying the graph.

A recent paper by Wilder and Sukthankar [29] formulates a sparsification method for social networks which aims to preserve the stationary distribution of random walks. Their method can be expressed as a nonlinear optimization problem with linear knapsack

constraints [6, 24]:

$$\text{(GSK-NL)} \qquad \max_{\tilde{E}\subseteq E} f(\tilde{E}) \quad \text{subject to} \quad \sum_{e\in\tilde{E}} c_e \le W.$$

Optimization problems of the form (GSK-NL) have been studied and applied to other problems related to network analysis. Leskovec et al. [18] studied the problem of selecting a subset of vertices in a network for sensor placement under budget constraints. This application could also be viewed as a vertex-filtering based graph sparsification problem.

3 A Fast Approximation Scheme

In this section, we discuss our implementation of a fast approximation algorithm for the 0/1 knapsack problem. Note that the sparsification methods discussed in the prior section – backbone extraction [23, 25], Local Similarity Filter [22], and Local Degree Filter [19] – are straightforward to implement and do not need to be solved under the knapsack formulation. We want to create a general sparsification framework that permits global or local edge centrality-based filtering, with the flexibility of introducing user-defined linear constraints.

Given integer profits and weights, the problem (GSK) can be solved exactly in pseudopolynomial time using a dynamic programming-based scheme [2, 7, 13]. However, this approach is prohibitively expensive for large graphs, and in our sparsification scenarios, it is not guaranteed that profits and weights are integers. Approximation algorithms for the 0/1 knapsack problem and variants have been studied extensively [1, 8, 11, 12, 14, 16, 20, 21]. There are two main kinds of approximation algorithms for knapsack: fully-polynomial time approximation schemes (FPTAS) that can achieve a result that is $1-\epsilon$ of the optimal value for all input error tolerance values ϵ [11, 12, 16, 21]; linear-time greedy approximation algorithms that have approximation quality guarantees as a fixed percentage of the optimal value [1, 8, 20]. The complexity of FPTAS algorithms is independent of the knapsack cost bound W, but polynomial in the number of items. In this section, we apply a greedy approximation algorithm based on [1, 14, 18] and explain how it can be implemented in $O(|E|)$ time. We assume that the profits $\{p_e|e \in E\}$ and the costs $\{c_e|e \in E\}$ are precomputed.

The commonly-used greedy approximation method for the knapsack problem [1, 8, 14] starts with an empty set $E_0^{\mathcal{F}}$, and expands $E_k^{\mathcal{F}}$ to $E_{k+1}^{\mathcal{F}} = E_k^{\mathcal{F}} \cup \{e_k^{\mathcal{F}}\}$ by adding the edge $e_k^{\mathcal{F}}$ that gives the maximal "efficiency",

$$e_k^{\mathcal{F}} = \max_{e\in E\setminus E_k^{\mathcal{F}}} \frac{p_e}{c_e},$$

until reaching the $k^{\mathcal{F}}$ such that $\sum_{i=0}^{k^{\mathcal{F}}-1} c_{e_i^{\mathcal{F}}} \le W < \sum_{i=0}^{k^{\mathcal{F}}} c_{e_i^{\mathcal{F}}}$. Selecting $\cup_{i=0}^{k^{\mathcal{F}}-1} e_i^{\mathcal{F}}$ or $\{e_{k^{\mathcal{F}}}^{\mathcal{F}}\}$ that corresponds to $\max\{\sum_{i=0}^{k^{\mathcal{F}}-1} p_{e_i^{\mathcal{F}}}, p_{e_{k^{\mathcal{F}}}^{\mathcal{F}}}\}$ gives a 2-approximation to the knapsack problem [1, 8, 14]. However, in the context of graph simplification formulation (GSK), if $p_{e_{k^{\mathcal{F}}}^{\mathcal{F}}} > \sum_{i=0}^{k^{\mathcal{F}}-1} p_{e_i^{\mathcal{F}}}$, constructing a simplified graph $\tilde{G} = (V, \{e_{k^{\mathcal{F}}}^{\mathcal{F}}\})$ with just a single edge $\{e_{k^{\mathcal{F}}}^{\mathcal{F}}\}$ may not be really usable in the context of graph sparsification. However, since the linear objective $f(\tilde{E}) = \sum_{e\in\tilde{E}} p_e$ is a submodular function with $f(E_A \cup \{e\}) - f(E_A) = f(E_B \cup \{e\}) - f(E_B) = p_e$, the greedy algorithm for submodular function maximization under knapsack constraints [15, 18] could be applied here. Analogous to expanding the edge set based on the greedy efficiency rule, the greedy profit rule initializes $E_0^{\mathcal{P}} = \emptyset$ and selects $e_k^{\mathcal{P}}$ in the k^{th} iteration that gives the largest profit:

$$e_k^{\mathcal{P}} = \max_{e\in E\setminus E_k^{\mathcal{P}}} p_e$$

until the iteration $k^{\mathcal{P}}$ such that adding $e_{k^{\mathcal{P}}}^{\mathcal{P}}$ will exceed the knapsack weight constraint in problem (GSK). Leskovec et al. [18] show that using $\max\{\sum_{i=0}^{k^{\mathcal{F}}-1} p_{e_i^{\mathcal{F}}}, \sum_{i=0}^{k^{\mathcal{P}}-1} p_{e_i^{\mathcal{P}}}\}$ results in a solution that is at least $\frac{1}{2}(1-1/e)$ of the optimal solution for general non-decreasing submodular functions. Hence, this result is applicable to problem (GSK) as well. The case of a single edge being retained is unlikely to occur for graphs we consider and when using edge centrality measures such as the Jaccard score.

The edge sets $\{e_i^{\mathcal{F}}\}_{i=0}^{k^{\mathcal{F}}-1}$ and $\{e_i^{\mathcal{P}}\}_{i=0}^{k^{\mathcal{P}}-1}$ can be determined by sorting $\{p_e\}_{e\in E}$ and $\{p_e/c_e\}_{e\in E}$. A faster method that avoids sorting is to use the weighted medians.Given a list of profits $\{p_e|e \in E\}$ and nonnegative costs $\{c_e|e \in E\}$, define the *weighted median profit* to be the profit $p^* \in \{p_e|e \in E\}$ such that

$$\sum_{e:p_e>p^*} c_e < W \le \sum_{e:p_e\ge p^*} c_e.$$

Similarly, let $f_e = p_e/c_e$, and define the *weighted median efficiency* to be the $f^* \in \{f_e|e \in E\}$ satisfying

$$\sum_{e:f_e>f^*} c_e < W \le \sum_{e:f_e\ge f^*} c_e$$

Using a slight modification of the selection sort algorithm, we can find the weighted medians p^* and f^* in $O(|E|)$ time [4, 7, 14]. After obtaining the weighted medians p^* and f^*, the set $\{e_i^{\mathcal{F}}\}_{i=0}^{k^{\mathcal{F}}-1}$ and $\{e_i^{\mathcal{P}}\}_{i=0}^{k^{\mathcal{P}}-1}$ of selected edges can be decided in two passes over the set of all edges. We describe this procedure for the greedy profit strategy. Let $E^{\mathcal{P}}$ denote the set of edges selected by the greedy profit rule. In one pass over E,

we partition the edges into $\{e \mid p_e > p^*\}$, $\{e \mid p_e = p^*\}$ and $\{e \mid p_e < p^*\}$. All edges in $\{e \mid p_e > p^*\}$ can be added safely to $E^{\mathcal{P}}$ and all edges in $\{e \mid p_e < p^*\}$ can be disregarded by the greedy rule. During the first pass, the remainder weight $R = W - \sum_{e:p_e>p^*} c_e$ is computed. The second pass over $\{e \mid p_e = p^*\}$ can be performed in arbitrary order until the remainder weight R is exhausted. Therefore, the running time is $O(|E|)$. The greedy efficiency rule can be implemented in a similar manner. The pseudocode for the full solver is provided in Algorithm 1.

Algorithm 1 Pseudocode for the GSK problem solver.

Input: $G(V, E)$, profits $\{p_e | e \in E\}$, costs $\{c_e | e \in E\}$, weight constraint W.
Output: $\tilde{E}$

$E^{\mathcal{P}} \leftarrow \emptyset$
$S(\mathcal{P}) \leftarrow 0$ ▷ greedy profit rule solution value
$R(\mathcal{P}) \leftarrow W$ ▷ remainder weight
Set p^* to computed weighted median profit
Initialize array $T(\mathcal{P})$
for $e \in E$ **do** ▷ $O(|E|)$
    if $p_e > p^*$ **then**
        add e to $E^{\mathcal{P}}$
        $S(\mathcal{P}) \leftarrow S(\mathcal{P}) + p_e$
        $R(\mathcal{P}) \leftarrow R(\mathcal{P}) - c_e$
    else if $p_e = p^*$ **then**
        add e to $T(\mathcal{P})$
add $e \in T(\mathcal{P})$ to $E^{\mathcal{P}}$ until $R(\mathcal{P})$ is exhausted
for $e \in E$ **do**
    Compute efficiency $f_e \leftarrow p_e/c_e$
$E^{\mathcal{F}} \leftarrow \emptyset$
$S(\mathcal{F}) \leftarrow 0$ ▷ greedy efficiency rule solution value
$R(\mathcal{F}) \leftarrow W$ ▷ remainder weight
Set f^* to computed weighted median efficiency
Initialize array $T(\mathcal{F})$
for $e \in E$ **do** ▷ $O(|E|)$
    if $f_e > f^*$ **then**
        add e to $E^{\mathcal{F}}$
        $S(\mathcal{F}) \leftarrow S(\mathcal{F}) + p_e$
        $R(\mathcal{F}) \leftarrow R(\mathcal{F}) - c_e$
    else if $f_e = f^*$ **then**
        add e to $T(\mathcal{F})$
add $e \in T(\mathcal{F})$ to $E^{\mathcal{F}}$ until $R(\mathcal{F})$ is exhausted
if $S(\mathcal{P}) \geq S(\mathcal{F})$ **then**
    return $E^{\mathcal{P}}$
else
    return $E^{\mathcal{F}}$

4 Constructing New Sparsification Schemes

We now discuss approaches to set the edge profits $\{p_e \mid e \in E\}$ and costs $\{c_e \mid e \in E\}$ in the GSK formulation. We have two goals in mind: First, the profits should attempt to preserve a structural property.

Algorithm 2 `DistributeCentrality`

Input: network $G(V, E)$, centrality $\{s(u)|u \in V\}$
Output: edge scores $\{s(u, v)|u \in V, \langle u, v\rangle \in E\}$

if G is undirected **then**
    for $e \in E$ **do**
        $u, v \leftarrow$ end points of e
        create directional edges $\langle u, v\rangle$ and $\langle v, u\rangle$
for $u \in V$ **do**
    for $\forall\langle u, v\rangle$ attached to u **do**
        $s(u, v) \leftarrow \left(d(v)/\sum_{\langle u,z\rangle\in E} d(z)\right) s(u)$

Second, we want to reuse precomputed vertex/edge centrality scores or other information available apriori in order to set the edge costs. For example, in case of online social network analysis, suppose the PageRank scores for each vertex have been already computed. How can we use this information to sparsify the graph and perform additional network analysis tasks? Ideally, we might wish that after sparsification, the set of top-ranked vertices using PageRank remains the same, and additionally other properties such as clustering coefficient distribution or community structure are also preserved. More generally, given a set of precomputed vertex centrality scores $\{s(u) \mid u \in V\}$, we devise a heuristic to construct a set of edge scores $\{s(u, v) \mid u \in V, \langle u, v\rangle \in E\}$. If our primary interest is to preserve the centrality rankings, then the edge scores could be used directly as profits. Otherwise, the edge scores could be transformed into costs, which would discriminate the edges based on their contribution to the vertex centralities in the original network.

We propose using a simple method for constructing edge scores that works on both directed and undirected graphs. For directed graphs, on each vertex $u \in V$, the centrality score $s(u)$ is distributed among its outgoing (or incoming) edges $\{\langle u, v\rangle \in E\}$. Each outgoing edge $e = \langle u, v\rangle$ obtains a fraction of $s(u)$, proportional to $d(v)/\sum_{\langle u,z\rangle\in E} d(z)$. The intuition for distributing the centrality scores on outgoing edges is that the influence of a vertex is more likely to propagate along its high-degree neighbors. For undirected graphs, we duplicate each edge $e \in E$ to create two directed links $\langle u, v\rangle$ and $\langle v, u\rangle$. This operation enlarges size of the knapsack decision variables from $|E|$ to $2|E|$. After solving problems (GSK), an edge e is retained in the simplified network if at least one of $\langle u, v\rangle$ and $\langle v, u\rangle$ is selected in the knapsack solution. We summarize this process of assigning edge scores in Algorithm 2.

Let $\texttt{DPR}_{ij}$ denote the score on edge $\langle i, j\rangle$ when Algorithm 2 is applied to generate edge scores based on PageRank. Since $\texttt{DPR}_{ij} \in (0, 1]$, $1 - \texttt{DPR}_{ij}$ could

be used as edge costs, so that a weak edge in the sense of contribution to PageRank has a large cost in problem (GSK). We extend the Jaccard score-based Local Similarity method and Global Similarity methods of Satuluri et al. [22] to use $(1 - \texttt{DPR}_{ij})$ as edge costs, and enforcing constraints

$$\sum_{i \in V} \sum_{\langle i,j \rangle \in \tilde{E}} (1 - \texttt{DPR}_{ij}) \leq W,$$

where W is a user-supplied parameter. W is set to be a percentage of $2|E|$, since $\sum_{i \in V} \sum_{\langle i,j \rangle \in E} (1 - \texttt{DPR}_{ij}) = 2|E| - 1$. We also devise a new method that uses $\texttt{DPR}_{ij}$ as the profit for edge $\langle i, j \rangle$, and enforces the unit-cost constraint $\sum_{i \in V} \sum_{\langle i,j \rangle \in \tilde{E}} 1 \leq W$. The motivation of this method is to preserve the vertices with top PageRank scores after sparsification.

Table 1: Graphs used in empirical evaluation.

Name	$\lvert V \rvert$ ($\times 10^6$)	$\lvert E \rvert$ ($\times 10^6$)
`com-Amazon` [30]	0.335	0.9
`com-DBLP` [30]	0.32	1.05
`com-Youtube` [30]	1.13	3
`roadNet-CA` [17]	1.97	5.53
`as-Skitter` [17]	1.7	11
`com-LiveJournal` [30]	4	34.68
`com-Orkut` [30]	3.07	117

5 Empirical Evaluation

In this section, we empirically evaluate six sparsification methods that can be expressed in the form of either problem (GSK) or its simpler unconstrained version. We used seven sparse graphs from the Stanford large network dataset collection SNAP [26], listed in Table 1. We implemented the approximate knapsack solution scheme in C+ and verified correctness with several test instances. The six sparsification methods considered are summarized in Tables 2 and 3. All experiments were run on a single server of Cyberstar, a Penn State compute cluster. The server we run our programs on is a dual-socket quad-core Intel Nehalem system (Intel Xeon X5550 processor) with 32 GB main memory.

5.1 Solver Performance. We evaluate performance of the solver for the problem (GSK). Unlike the exact pseudopolynomial dynamic programming approach, the running time of Algorithm 1 is independent of the user-supplied weight parameter W and the actual settings for the edge profits $\{p_e | e \in E\}$ and costs $\{c_e | e \in E\}$. We computed the execution time of the solver for the seven graphs in Table 1. The graph sizes differ by nearly

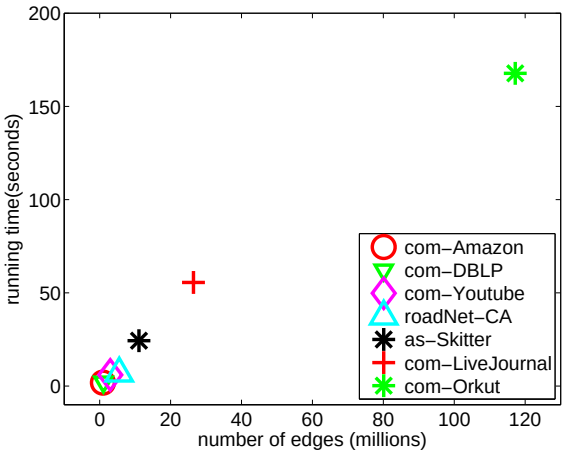


Figure 1: Solver running time on various graphs.

two orders of magnitude, going from `com-Amazon` to `com-Orkut`. Figure 1 gives the running times. As the number of graph edges increases, the execution time appears to increase linearly. This empirical evaluation is one way of demonstrating the practical efficacy of the solver. Note that this figure only gives the solver execution time, and does not include the time taken to compute profits and costs. Our current implementations for PageRank and Jaccard scores are straightforward and untuned. For `com-DBLP`, the running times of JLPR for PageRank computation (costs), Jaccard score computation (profits), and the greedy approximation are 0.21, 2.28, and 2.1 seconds, respectively.

5.2 Comparing Sparsification Methods. We now apply the solver to the six problem formulations in Tables 2 and 3. In addition, we sparsify the graphs using uniform random edge (RE) sampling as a baseline. The abbreviations used in the figures correspond to the methods in Tables 2 and 3. The knapsack bound W is user-supplied. We set the problem parameters to achieve various sparsification ratios of $|\tilde{E}|/|E|$. We use three structural properties to evaluate the sparsification schemes: vertex degree distribution, top-ranked PageRank vertices and average clustering coefficient. We use Spearman's rank correlation coefficient [27] to compute correlation between vertex degree rankings in the sparsified graph and in the original graph. This value will be close to 1 if the degree rankings are highly correlated. For PageRank, we evaluate the methods based on the proportion of the top 10% PageRank-ordered vertices in the original graph that are preserved after sparsification. We report experimental results on `com-Amazon`, `com-DBLP`, `com-Youtube` and `com-LiveJournal`.

Table 2: Problem (GSK)-based graph sparsification strategies used in empirical evaluation.

Label	$p(i,j)$	*Constraint*	*Motivation*
JLPR	$\frac{1}{d_i}\sum_{\{k\mid\langle i,k\rangle\in\tilde{E}\}}\mathbf{1}\{\texttt{Jac}_{ik}\leq\texttt{Jac}_{ij}\}$	$\sum_{i\in V}\sum_{\langle i,j\rangle\in\tilde{E}}(1-\texttt{DPR}_{ij})\leq W$	local similarity + PageRank
JGPR	$\texttt{Jac}_{ij}$	$\sum_{i\in V}\sum_{\langle i,j\rangle\in\tilde{E}}(1-\texttt{DPR}_{ij})\leq W$	similarity + PageRank
PR	$\texttt{DPR}_{ij}$	$\sum_{i\in V}\sum_{\langle i,j\rangle\in\tilde{E}}1\leq W$	PageRank
JG	$\texttt{Jac}_{ij}$	$\sum_{i\in V}\sum_{\langle i,j\rangle\in\tilde{E}}1\leq W$	similarity [22]

Table 3: Unconstrained sparsification strategies used in empirical evaluation.

Label	$p(i,j)$	*Motivation*
DL	retain top $\lceil d_i^\alpha\rceil$ edges ranked by $\texttt{Deg}_{ij}$ locally	hub vertices [19]
JL	retain top $\lceil d_i^\alpha\rceil$ edges ranked by $\texttt{Jac}_{ij}$ locally	local similarity [22]

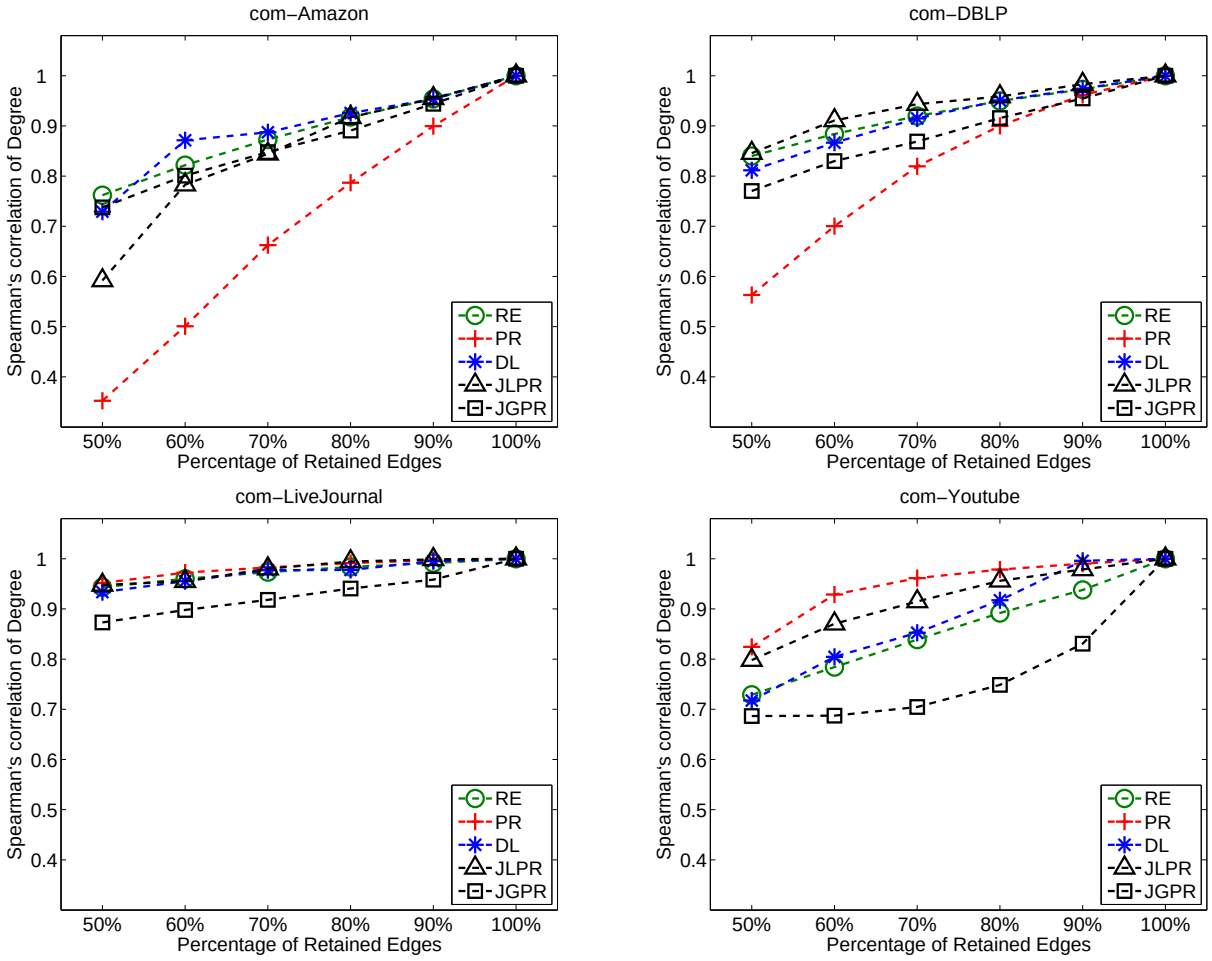


Figure 2: Spearman's rank coefficient of degree for the original graph and sparsified graph, at various sparsification thresholds.

Figure 2 plots the Spearman's correlation for vertex degree rankings at various sparsification ratios. There is no single method that consistently outperforms others. In [19], the authors perform a similar experiment, for vertex degree ranking using Spearman's rank correlation coefficient, over a large collection of social net-

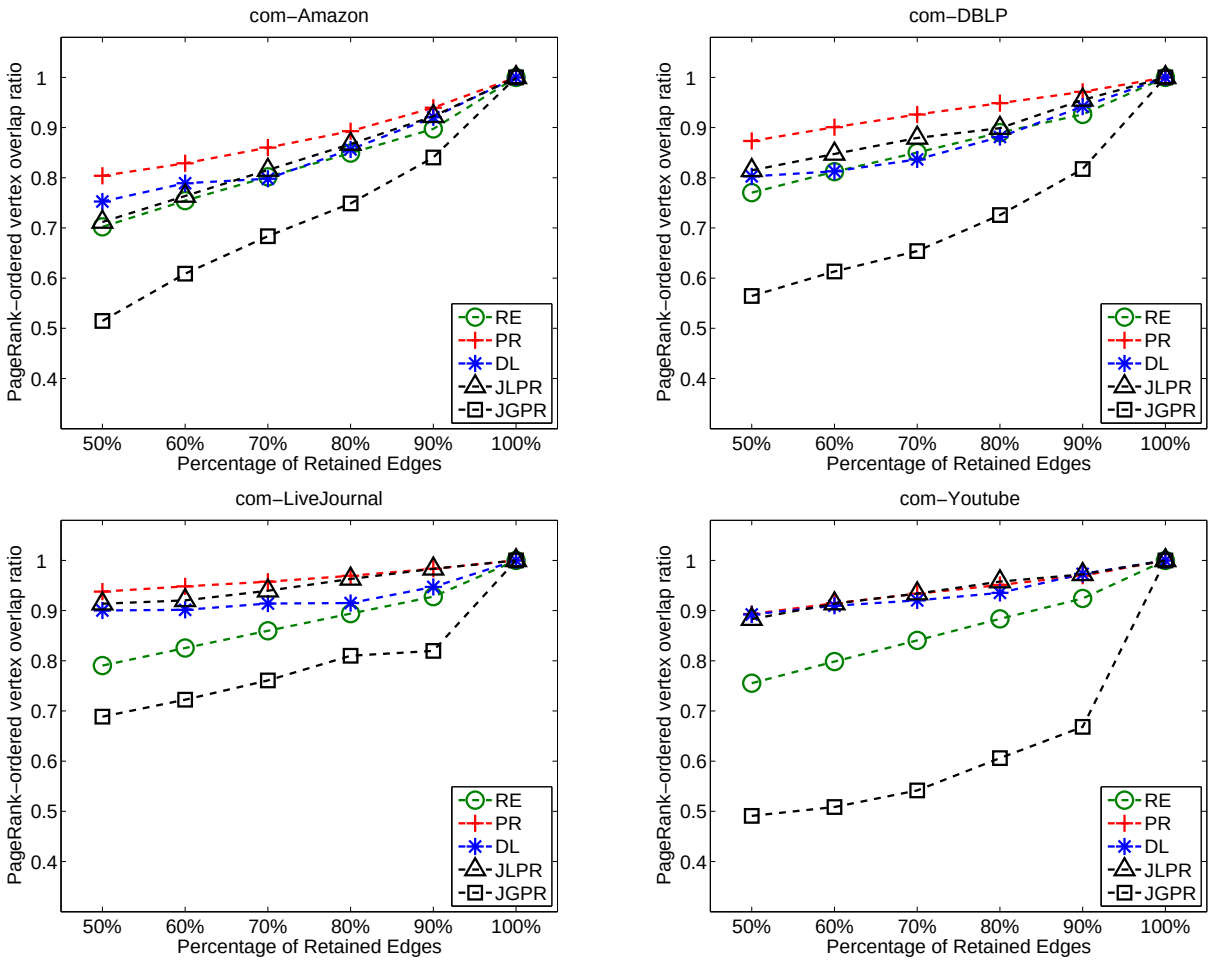


Figure 3: The fraction of vertices that overlap in the top 10% PageRank-ordered vertex sets in the original and the sparsified graph, at various sparsification thresholds.

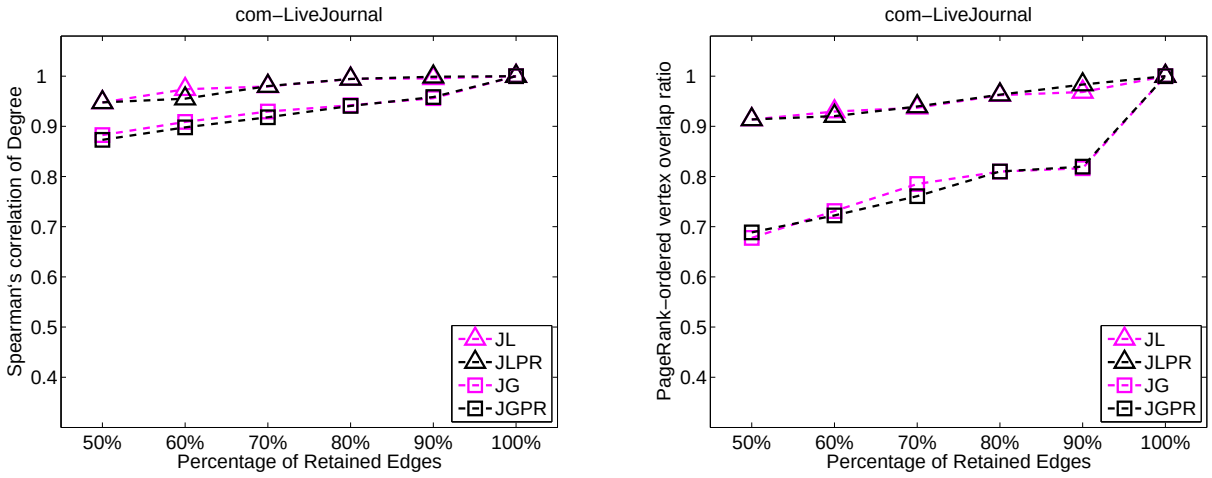


Figure 4: Comparing Jaccard similarity-based sparsifier performance on the `com-LiveJournal` graph.

works. Their results show that DL and RE consistently outperform JL. In our experiments, RE, PR, DL, and JLPR all perform qualitatively similarly on the social network `com-LiveJournal`. However, the performance diverges on the other graphs. PR's performance is also noteworthy: it performs significantly worse than RE for `com-Amazon` and `com-DBLP`, but better than all methods for `com-Youtube`.

Results showing the overlap ratio of top 10% PageRank-ordered vertices after applying a sparsification method are shown in Figure 3. As expected, PR consistently outperforms the other methods, since in PR, the edge profit `DPR` is based on the vertex PageRank scores. DL and JLPR give similar results for the overlap ratio in all four graphs, and slightly underperform PR. JGPR sparsification significantly distorts the top 10% PageRank-ordered vertex set in all experiments.

The comparison of JL, JLPR, JG, and JGPR on the Spearman's correlation of degrees and the overlap ratio of top 10% PageRank vertices are displayed in Figure 4. For all the graphs, the difference between JL and JLPR, and the difference between JG and JGPR is very small. Hence we do not show JL and JG results in Figures 2 and 3.

We report the deviation of average clustering coefficient Ave-CC from the original graph after sparsification on `com-DBLP` and `com-Amazon` in Figure 5. The deviation is computed as Ave-CC($\tilde{G}$)−Ave-CC(G). The average clustering coefficients for `com-DBLP` and `com-Amazon` are 0.6324 and 0.3967, respectively. These two graphs have the largest average clustering coefficient among the ones listed in Table 1. Surprisingly, PR causes the least deviation of average clustering coefficient on `com-DBLP`. JLPR and JGPR, which aim to preserve intra-cluster edges, produce positive deviations at some sparsification thresholds on `com-Amazon`. RE reduces the average clustering coefficient linearly on decreasing the number of retained edges, an observation also pointed out in [19].

6 Conclusions and Future Work

In this work, we explore using the knapsack problem for graph edge sparsification. Sparsifying large graphs aids in graph visualization and serves as a speedup technique for graph computations such as community detection and centrality analysis. Our proposed knapsack-based sparsification permits both global and local centrality-based edge filtering, and additionally lets us specify fine-grained linear constraints. We implement a greedy linear-time approximation solution scheme for this problem. Our preliminary empirical evaluation looks at two new formulations and compares them to known edge sparsification strategies using three criteria. We plan to perform a detailed empirical evaluation in future work, by expanding the collection of input graphs, the evaluation criteria (studying sparsification impact on other topological properties; studying impact of greedy solver on sparsification quality), and the methods chosen (include advanced sampling-based methods, as well as other new knapsack problem combinations). We will also attempt to identify cases where a general nonlinear submodular function can be approximated as a piecewise linear function. For cases where such approximations are possible, our knapsack solver can be readily used. We also hope to demonstrate, in future work, use of our sparsification schemes for scalable graph visualization and as a general speedup strategy.

Acknowledgments

We thank the reviewers for their thoughtful comments and suggestions to improve the paper. This work was supported in part through instrumentation funded by the National Science Foundation through grant OCI-0821527. This work is also supported by National Science Foundation grants ACI-1253881 and CCF-1439057.

References

[1] E. Balas and E. Zemel. An algorithm for large zero-one knapsack problems. *operations Research*, 28(5):1130–1154, 1980.

[2] R. Bellman. Notes on the theory of dynamic programming iv-maximization over discrete sets. *Naval Research Logistics Quarterly*, 3(1-2):67–70, 1956.

[3] A. A. Benczúr and D. R. Karger. Randomized approximation schemes for cuts and flows in capacitated graphs. *SIAM Journal on Computing*, 44(2):290–319, 2015.

[4] M. Blum, R. W. Floyd, V. Pratt, R. L. Rivest, and R. E. Tarjan. Time bounds for selection. *Journal of computer and system sciences*, 7(4):448–461, 1973.

[5] F. Bonchi, G. D. F. Morales, A. Gionis, and A. Ukkonen. Activity preserving graph simplification. *Data Mining and Knowledge Discovery*, 27(3):321–343, 2013.

[6] K. M. Bretthauer and B. Shetty. The nonlinear knapsack problem–algorithms and applications. *European Journal of Operational Research*, 138(3):459–472, 2002.

[7] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms (3. ed.)*. MIT Press, 2009.

[8] D. Fayard and G. Plateau. An algorithm for the solution of the 0–1 knapsack problem. *Computing*, 28(3):269–287, 1982.

[9] N. J. Foti, J. M. Hughes, and D. N. Rockmore. Nonparametric sparsification of complex multiscale networks. *PLoS ONE*, 6(2):e16431, 2011.

[10] W. S. Fung, R. Hariharan, N. J. Harvey, and D. Panigrahi. A general framework for graph sparsification.

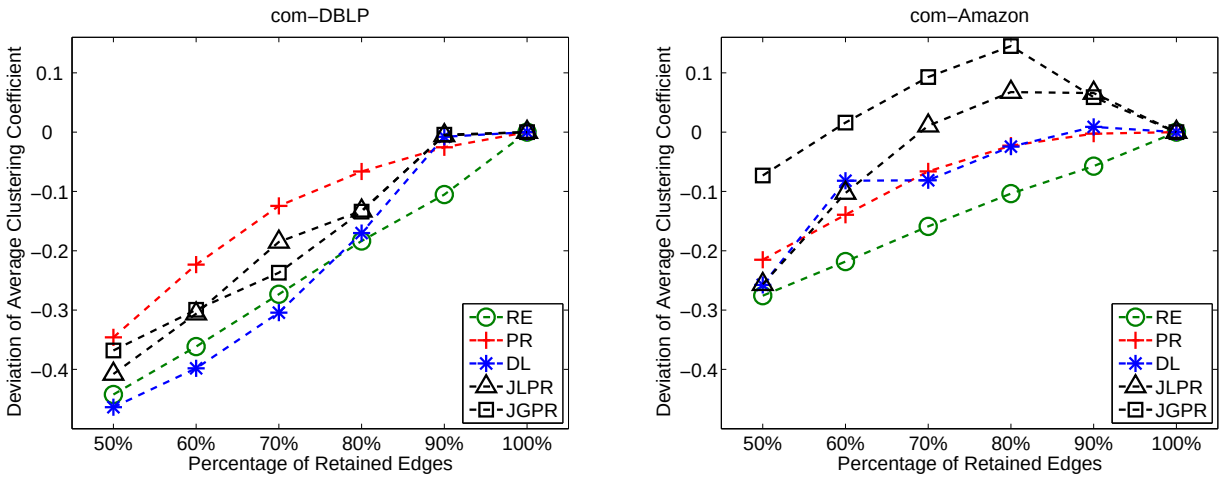


Figure 5: Deviation of average clustering coefficient at various sparsification thresholds.

In *Proc. 43rd Annual ACM Symp. on Theory of Computing (STOC)*, pages 71–80. ACM, 2011.

[11] O. H. Ibarra and C. E. Kim. Fast approximation algorithms for the knapsack and sum of subset problems. *Journal of the ACM*, 22(4):463–468, 1975.

[12] H. Kellerer and U. Pferschy. A new fully polynomial time approximation scheme for the knapsack problem. *Journal of Combinatorial Optimization*, 3(1):59–71, 1999.

[13] J. M. Kleinberg and Éva Tardos. *Algorithm Design*. First edition, 2006.

[14] B. Korte and J. Vygen. *Combinatorial optimization: Theory and algorithms*. Fourth edition, 2008.

[15] A. Krause and D. Golovin. Submodular function maximization. *Tractability: Practical Approaches to Hard Problems*, 3:19, 2012.

[16] E. L. Lawler. Fast approximation algorithms for knapsack problems. *Mathematics of Operations Research*, 4(4):339–356, 1979.

[17] J. Leskovec, J. Kleinberg, and C. Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proc. 11th ACM SIGKDD Int'l. Conf. on Knowledge discovery and data mining (KDD)*, pages 177–187. ACM, 2005.

[18] J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, J. VanBriesen, and N. Glance. Cost-effective outbreak detection in networks. In *Proc. 13th ACM SIGKDD Int'l. Conf. on Knowledge discovery and data mining (KDD)*, pages 420–429. ACM, 2007.

[19] G. Lindner, C. L. Staudt, M. Hamann, H. Meyerhenke, and D. Wagne. Structure-preserving sparsification of social networks. In *Proc. IEEE/ACM Int'l. Conf. on Advances in Social Networks Analysis and Mining (ASONAM)*, pages 448–454. IEEE/ACM, 2015.

[20] D. Pisinger. Linear time algorithms for knapsack problems with bounded weights. *Journal of Algorithms*, 33(1):1–14, 1999.

[21] S. Sahni. Approximate algorithms for the 0/1 knapsack problem. *Journal of the ACM*, 22(1):115–124, 1975.

[22] V. Satuluri, S. Parthasarathy, and Y. Ruan. Local graph sparsification for scalable clustering. In *Proc. ACM SIGMOD Int'l. Conf. on Management of Data (SIGMOD)*, pages 721–732. ACM, 2011.

[23] M. Á. Serrano, M. Boguná, and A. Vespignani. Extracting the multiscale backbone of complex weighted networks. *Proceedings of the National Academy of Sciences (PNAS)*, 106(16):6483–6488, 2009.

[24] T. C. Sharkey, H. E. Romeijn, and J. Geunes. A class of nonlinear nonseparable continuous knapsack and multiple-choice knapsack problems. *Mathematical Programming*, 126(1):69–96, 2011.

[25] P. B. Slater. Multiscale network reduction methodologies: Bistochastic and disparity filtering of human migration flows between 3,000+ US counties. *arXiv preprint arXiv:0907.2393*, 2009.

[26] SNAP Stanford large network dataset collection. `http://snap.stanford.edu/data/index.html`, last accessed Jan 2016.

[27] C. Spearman. The proof and measurement of association between two things. *The American Journal of Psychology*, 15(1):72–101, 1904.

[28] D. A. Spielman and S.-H. Teng. Spectral sparsification of graphs. *SIAM Journal on Computing*, 40(4):981–1025, 2011.

[29] B. Wilder and G. Sukthankar. Sparsification of social networks using random walks. In *Proc. 8th ASE Int'l. Conf. on Social Computation (SocialCom)*. ASE, 2015.

[30] J. Yang and J. Leskovec. Defining and evaluating network communities based on ground-truth. *Knowledge and Information Systems*, 42(1):181–213, 2015.

Trust from the past: Bayesian Personalized Ranking based Link Prediction in Knowledge Graphs

Baichuan Zhang* Sutanay Choudhury† Mohammad Al Hasan‡ Xia Ning‡
Khushbu Agarwal† Sumit Purohit† Paola Pesantez Cabrera§

Abstract

Link prediction, or predicting the likelihood of a link in a knowledge graph based on its existing state is a key research task. It differs from a traditional link prediction task in that the links in a knowledge graph are categorized into different predicates and the link prediction performance of different predicates in a knowledge graph generally varies widely. In this work, we propose a latent feature embedding based link prediction model which considers the prediction task for each predicate disjointly. To learn the model parameters it utilizes a Bayesian personalized ranking based optimization technique. Experimental results on large-scale knowledge bases such as YAGO2 show that our link prediction approach achieves substantially higher performance than several state-of-art approaches. We also show that for a given predicate the topological properties of the knowledge graph induced by the given predicate edges are key indicators of the link prediction performance of that predicate in the knowledge graph.

1 Introduction

A knowledge graph is a repository of information about entities, where entities can be any thing of interest such as people, location, organization or even scientific topics, concepts, etc. An entity is frequently characterized by its association with other entities. As an example, capturing the knowledge about a company involves listing its products, location and key individuals. Similarly, knowledge about a person involves her name, date and place of birth, affiliation with organizations, etc. Resource Description Framework (RDF) is a frequent choice for capturing the interactions between two entities. A RDF dataset is equivalent to a heterogeneous graph, where each vertex and edge can belong to different classes. The class information captures taxonomic hierarchies between the type of various entities and relations. As an example, a knowledge graph may identify Kobe Bryant as a basketball player, while its ontology will indicate that a basketball player is a particular type of athlete. Thus, one will be able to query for famous athletes in the United States and find Kobe Bryant.

The past few years have seen a surge in research on knowledge representations and algorithms for building knowledge graphs. For example, Google Knowledge Vault [6], and IBM Watson [9] are comprehensive knowledge bases which are built in order to answer questions from the general population. As evident from these works, it requires multitude of efforts to build a domain specific knowledge graph, which are, triple extraction from nature language text, entity and relationship mapping [25], and link prediction [21]. Specifically, triples extracted from the text data sources using state of the art techniques such as OpenIE [8] and semantic role labeling [5] are extremely noisy, and simply adding noisy triple facts into knowledge graph destroys its purpose. So computational methods must be devised for deciding which of the extracted triples are worthy of insertion into a knowledge graph. There are several considerations for this decision making: (1) trustworthiness of the data sources; (2) a belief value reported by a natural language processing engine expressing its confidence in the correctness of parsing; and (3) prior knowledge of subjects and objects. This particular work is motivated by the third factor.

Link prediction in knowledge graph is simply a machine learning approach for utilizing prior knowledge of subjects and objects as available in the knowledge graph for estimating the confidence of a candidate triple. Consider the following example: given a social media post "I wish Tom Cruise was the president of United States", a natural language processing engine will extract a triple

*Department of Computer and Information Science, Indiana University - Purdue University Indianapolis, USA, bz3@umail.iu.edu. The work was conducted during author's internship at Pacific Northwest National Laboratory

†Pacific Northwest National Laboratory, Richland, WA, USA, {Sutanay.Choudhury,Khushbu.Agarwal,Sumit.Purohit}@pnnl.gov

‡Department of Computer and Information Science, Indiana University - Purdue University Indianapolis, USA, {alhasan,xning}@cs.iupui.edu

§Department of Computer Science, Washington State University, Pullman, WA, USA, p.pesantezcabrera@email.wsu.edu

("Tom Cruise", "president of", "United States"). On the other hand, a web crawler may find the fact that "Tom Cruise is president of Downtown Medical", resulting in the triple ("Tom Cruise", "president of", "Downtown Medical"). Although we generally do not have any information about the trustworthiness of the sources, our prior knowledge of the entities mentioned in this triples will enable us to decide that the first of the above triples is possibly wrong. Link prediction provides a principles approach for such a decision-making. Also note that, once we decide to add a triple to the knowledge graph, it is important to have a confidence value associated with it.

As we use a machine learning approach to compute the confidence of triple facts, it is important that we quantitatively understand the degree of accuracy of our prediction [28]. It is important, because for the same knowledge graph the prediction accuracy level varies from predicate to predicate. As an example, predicting one's school or workplace can be a much harder task than predicting one's liking for a local restaurant. Therefore, given two predicates "worksAt" and "likes", we expect to see widely varying accuracy levels. Also, the average accuracy levels vary widely from one knowledge graph to another. The desire to obtain a quantitative grasp on prediction accuracy is complicated by a number of reasons: 1) Knowledge graphs constructed from web text or using machine reading approaches can have a very large number of predicates that make manual verification difficult [6]; 2) Creation of predicates, or the resultant graph structure is strongly shaped by the ontology, and the conversion process used to generate RDF statements from a logical record in the data. Therefore, same data source can be represented in very different models and this leads to different accuracy levels for the same predicate. 3) The effectiveness of knowledge graphs have inspired their construction from every imaginable data source: product inventories (at retailers such as Wal-mart), online social networks (such as Facebook), and web pages (Google's Knowledge Vault). As we move from one data source to another, it is critical to understand what accuracy levels we can expect from a given predicate.

In this paper, we use a link prediction [1] approach for computing the confidence of a triple from the prior knowledge about its subject and object. Many works exist for link prediction [11] in social network analysis [4], but they differ from the link prediction in knowledge graph; for earlier, all the links are semantically similar, but for the latter based on the predicates the semantic of the links differs widely. So, existing link prediction methods are not very suitable for this task. We build our link prediction method by borrowing solutions from recommender system research which accept a user-item matrix and for a given user-item pair, they return a score indicating the likelihood of the user purchasing the item. Likewise, for a given predicate, we consider the set of subjects and objects as a user-item matrix and produce a real-valued score to measure the confidence of the given triple. For training the model we use Bayesian personalized ranking (BPR) based embedding model [23], which has been a major work in the recommendation system. In addition, we also study the performance of our proposed link prediction algorithm in terms of topological properties of knowledge graph and present a linear regression model to reason about its expected level of accuracy for each predicate.

Our contributions in this work are outlined below:

1. We implement a Link Prediction approach for estimating confidence for triples in a Knowledge Graph. Specifically, we borrow from successful approaches in the recommender systems domain, adopt the algorithms for knowledge graphs and perform a thorough evaluation on a prominent benchmark dataset.

2. We propose a Latent Feature Embedding based link recommendation model for prediction task and utilize Bayesian Personalized Ranking based optimization technique for learning models for each predicate (Section 4). Our experiments on the well known YAGO2 knowledge graph (Section 5) show that the BPR approach outperforms other competing approaches for a significant set of predicates (Figure 1).

3. We apply a linear regression model to quantitatively analyze the correlation between the prediction accuracy for each predicate and the topological structure of the induced subgraph of the original Knowledge Graph. Our studies show that metrics such as clustering coefficient or average degree can be used to reason about the expected level of prediction accuracy (Section 5.3, Figure 2).

2 Related Work

There is a large body of work on link prediction in knowledge graph. In terms of methodology, factorization based and related latent variable models [3, 7, 13, 22, 25], graphical model [14], and graph feature based method [17, 18] are considered.

There exists large number of works which focus on factorization based models. The common thread

[1] We use link prediction and link recommendation interchangeably.

among the factorization methods is that they explain the triples via latent features of entities. [2] presents a tensor based model that decomposes each entity and predicate in knowledge graphs as a low dimensional vector. However, such a method fails to consider the symmetry property of the tensor. In order to solve this issue, [22] proposes a relational latent feature model, RESCAL, an efficient approach which uses a tensor factorization model that takes the inherent structure of relational data into account. By leveraging relational domain knowledge about entity type information, [3] proposes a tensor decomposition approach for relation extraction in knowledge base which is highly efficient in terms of time complexity. In addition, various other latent variable models, such as neural network based methods [6, 27], have been explored for link prediction task. However, the major drawback of neural network based models is their complexity and computational cost in model training and parameter tuning. Many of these models require tuning large number of parameters, thus finding the right combination of these parameters is often considered more of an art than science.

Recently graphical models, such as Probabilistic Relational Models [10], Relational Markov Network [29], Markov Logic Network [14, 24] have also been used for link prediction in knowledge graph. For instance, [24] proposes a Markov Logic Network (MLN) based approach, which is a template language for defining potential functions on knowledge graph by logical formula. Despite its utility for modeling knowledge graph, issues such as rule learning difficulty, tractability problem, and parameter estimation pose implementation challenge for MLNs.

Graph feature based approaches assume that the existence of an edge can be predicted by extracting features from the observed edges in the graph. Lao and Cohen [17,18] propose Path Ranking Algorithm (PRA) to perform random walk on the graph and compute the probability of each path. The main idea of PRA is to use these path probabilities as supervised features for each entity pair, and use any favorable classification model, such as logistic regression and SVM, to predict the probability of missing edge between an entity pair in a knowledge graph.

It has been demonstrated [1] that no single approach emerges as a clear winner. Instead, the merits of factorization models and graph feature models are often complementary with each other. Thus combining the advantages of different approaches for learning knowledge graph is a promising option. For instance, [20] proposes to use additive model, which is a linear combination between RESCAL and PRA. The combination results in not only decrease the training time but also increase the accuracy. [15] combines a latent feature model with an additive term to learn from latent and neighborhood-based information on multi-relational data. [6] fuses the outputs of PRA and neural network model as features for training a binary classifier. Our work strongly aligns with this combination approach. In this work, we build matrix factorization based techniques that have been proved successful for recommender systems and plan to incorporate graph based features in future work.

3 Background and Problem Statement

Definition 3.1. We define the knowledge graph as a collection of triple facts $G = (S, P, O)$, where $s \in S$ and $o \in O$ are the set of subject and object entities and $p \in P$ is the set of predicates or relations between them. $G(s, p, o) = 1$ if there is a direct link of type p from s to o, and $G(s, p, o) = 0$ otherwise.

Each triple fact in knowledge graph is a statement interpreted as "A relationship p holds between entities s and o". For instance, the statement "Kobe Bryant is a player of LA Lakers" can be expressed by the following triple fact ("Kobe Bryant", "playsFor", "LA Lakers").

Definition 3.2. For each relation $p \in P$, we define $G_p(S_p, O_p)$ as a bipartite subgraph of G, where the corresponding set of entities $s_p \in S_p$, $o_p \in O_p$ are connected by relation p, namely $G_p(s_p, o_p) = 1$.

Problem Statement: For every predicate $p \in P$ and given an entity pair (s, o) in G_p, our goal is to learn a link recommendation model M_p such that $x_{s,o} = M_p(s, o)$ is a real-valued score.

Due to the fact that the produced real-valued score is not normalized, we compute the probability $Pr(y^p_{s,o} = 1)$, where $y^p_{s,o}$ is a binary random variable that is true iff $G_p(s, o) = 1$. We estimate this probability Pr using the logistic function as follows:

$$Pr(y^p_{s,o} = 1) = \frac{1}{1 + exp(-x_{s,o})} \tag{3.1}$$

Thus we interpret $Pr(y^p_{s,o} = 1)$ as the probability that a vertex (or subject) s in the knowledge graph G is in a relationship of given type p with another vertex (or the object) o.

4 Methods

In this section, we describe our model, namely Latent Feature Embedding Model with Bayesian Personalized Ranking (BPR) based optimization technique that we propose for the task of link prediction in a knowledge graph. In our link prediction setting, for a given predicate p, we first construct its bipartite subgraph

$G_p(S_p, O_p)$. Then we learn the optimal low dimensional embeddings for its corresponding subject and object entities $s_p \in S_p$, $o_p \in O_p$ by maximizing a ranking based distance function. The learning process relies on Stochastic Gradient Descent (SGD). The SGD based optimization technique iteratively updates the low dimensional representation of s_p and o_p until convergence. Then the learned model is used for ranking the unobserved triple facts in descending order such that triple facts with higher score values have a higher probability of being correct.

4.1 Latent Feature Based Embedding Model

For each predicate p, the model maps both its corresponding subject and object entites s_p and o_p into low-dimensional continuous vector spaces, say $U_s^p \in \mathbb{R}^{1\times K}$ and $V_o^p \in \mathbb{R}^{1\times K}$ respectively. We measure the compatibility between subject s_p and object o_p as dot product of its corresponding latent vectors which is given as below:

$$x_{s_p,o_p} = (U_s^p)(V_o^p)^T + b_o^p \tag{4.2}$$

where $U^p \in \mathbb{R}^{|S|\times K}$, $V^p \in \mathbb{R}^{|O|\times K}$, and $b^p \in \mathbb{R}^{|O|\times 1}$. $|S|$ and $|O|$ denote the size of subject and object associated with predicate p respectively. K is the number of latent dimensions and $b_o^p \in \mathbb{R}$ is a bias term associated with object o. Given predicate p, the higher the score of x_{s_p,o_p}, the more similar the entities s_p and o_p in the embedded low dimensional space, and the higher the confidence to include this triple fact into knowledge base.

4.2 Bayesian Personalized Ranking

In collaborative filtering, positive-only data is known as implicit feedback/binary feedback. For example, in the eCommerce platform, some users only buy but do not rate items. Motivated by [23], we employ Bayesian Personalized Ranking (BPR) based approach for model learning. Specifically, in recommender system domain, given user-item matrix, BPR based approach assigns the preference of user for purchased item with higher score than un-purchased item. Likewise, under this context, we assign observed triple facts higher score than unobserved triple facts in knowledge base. We assume that unobserved facts are not necessarily negative, rather they are "less preferable" than the observed ones.

For our task, in each predicate p, we denote the observed subject/object entity pair as (s_p, o_p^+) and unobserved one as (s_p, o_p^-). The observed facts in our case are the existing link between s_p and o_p given G_p and unobserved ones are the missing link between them. Given this fact, BPR maximizes the following ranking based distance function:

$$BPR = \max_{\Theta_p} \sum_{(s_p,o_p^+,o_p^-)\in D_p} \ln \sigma(x_{s_p,o_p^+} - x_{s_p,o_p^-}) - \lambda_{\Theta_p} \mid\mid \Theta_p \mid\mid^2 \tag{4.3}$$

where D_p is a set of samples generated from the training data for predicate p, $G_p(s_p, o_p^+) = 1$ and $G_p(s_p, o_p^-) = 0$. And x_{s_p,o_p^+} and x_{s_p,o_p^-} are the predicted scores of subject s_p on objects o_p^+ and o_p^- respectively. We use the proposed latent feature based embedding model shown in Equation 4.2 to compute x_{s_p,o_p^+} and x_{s_p,o_p^-} respectively. The last term in Equation 4.3 is a l_2-norm regularization term used for model parameters $\Theta_p = \{U^p, V^p, b^p\}$ to avoid overfitting in the learning process. In addition, the logistic function $\sigma(.)$ in Equation 4.3 is defined as $\sigma(x) = \frac{1}{1+e^{-x}}$.

Notice that the Equation 4.3 is differentiable, thus we employ the widely used SGD to maximize the objective. In particular, at each iteration, for given predicate p, we sample one observed entity pair (s_p, o_p^+) and one unobserved one (s_p, o_p^-) using uniform sampling technique. Then we iteratively update the model parameters Θ_p based on the sampled pairs. Specifically, for each training instance, we compute the derivative and update the corresponding parameters Θ_p by walking along the ascending gradient direction.

For each predicate p, given a training triple (s_p, o_p^+, o_p^-), the gradient of BPR objective in Equation 4.3 with respect to U_s^p, $V_{o^+}^p$, $V_{o^-}^p$, $b_{o^+}^p$, $b_{o^-}^p$ can be computed as follows:

$$\begin{aligned}
\frac{\partial BPR}{\partial U_s^p} &= \frac{\partial \ln \sigma(x_{s_p,o_p^+} - x_{s_p,o_p^-})}{\partial U_s^p} - 2\lambda_s^p U_s^p \\
&= \frac{\partial \ln \sigma(x_{s_p,o_p^+} - x_{s_p,o_p^-})}{\partial \sigma(x_{s_p,o_p^+} - x_{s_p,o_p^-})} \times \frac{\partial \sigma(x_{s_p,o_p^+} - x_{s_p,o_p^-})}{\partial (x_{s_p,o_p^+} - x_{s_p,o_p^-})} \\
&\quad \times \frac{\partial (x_{s_p,o_p^+} - x_{s_p,o_p^-})}{\partial U_s^p} - 2\lambda_s^p U_s^p \\
&= \frac{1}{\sigma(x_{s_p,o_p^+} - x_{s_p,o_p^-})} \times \sigma(x_{s_p,o_p^+} - x_{s_p,o_p^-}) \\
&\quad \big(1 - \sigma(x_{s_p,o_p^+} - x_{s_p,o_p^-})\big) \times (V_{o^+}^p - V_{o^-}^p) - 2\lambda_s^p U_s^p \\
&= \big(1 - \sigma(x_{s_p,o_p^+} - x_{s_p,o_p^-})\big)(V_{o^+}^p - V_{o^-}^p) - 2\lambda_s^p U_s^p
\end{aligned} \tag{4.4}$$

We obtain the following using similar chain rule derivation.

$$\frac{\partial BPR}{\partial V_{o^+}^p} = \big(1 - \sigma(x_{s_p,o_p^+} - x_{s_p,o_p^-})\big) \times U_s^p - 2\lambda_{o^+}^p V_{o^+}^p \tag{4.5}$$

$$\frac{\partial BPR}{\partial V_{o^-}^p} = \big(1 - \sigma(x_{s_p,o_p^+} - x_{s_p,o_p^-})\big) \times (-U_s^p) - 2\lambda_{o^-}^p V_{o^-}^p \tag{4.6}$$

$$\frac{\partial BPR}{\partial b^p_{o^+}} = \big(1 - \sigma(x_{s_p,o^+_p} - x_{s_p,o^-_p})\big) \times 1 - 2\lambda^p_{o^+} b^p_{o^+} \tag{4.7}$$

$$\frac{\partial BPR}{\partial b^p_{o^-}} = \big(1 - \sigma(x_{s_p,o^+_p} - x_{s_p,o^-_p})\big) \times (-1) - 2\lambda^p_{o^-} b^p_{o^-} \tag{4.8}$$

Next, the parameters are updated as follows:

$$U^p_s = U^p_s + \alpha \times \frac{\partial BPR}{\partial U^p_s} \tag{4.9}$$

$$V^p_{o^+} = V^p_{o^+} + \alpha \times \frac{\partial BPR}{\partial V^p_{o^+}} \tag{4.10}$$

$$V^p_{o^-} = V^p_{o^-} + \alpha \times \frac{\partial BPR}{\partial V^p_{o^-}} \tag{4.11}$$

$$b^p_{o^+} = b^p_{o^+} + \alpha \times \frac{\partial BPR}{\partial b^p_{o^+}} \tag{4.12}$$

$$b^p_{o^-} = b^p_{o^-} + \alpha \times \frac{\partial BPR}{\partial b^p_{o^-}} \tag{4.13}$$

where α is the learning rate.

4.3 Pseudo-code and Complexity Analysis

The pseudo-code of our proposed link prediction model is described in Algorithm 1. It takes the knowledge graph G and a specific target predicate p as input and generates the low dimensional latent matrices U^p, V^p, b^p as output. Line 1 constucts the bipartite subgraph of predicate p, G_p given entire knowledge graph G. Line 2-3 compute the number of subject and object entities as m and n in resultant bipartite subgraph G_p respectively. Line 4 generates a collection of triple samples using uniform sampling technique. Line 5-7 initialize the matrices U^p, V^p, b^p using Gaussian distribution with 0 mean and 0.1 standard deviation, assuming all the entries in U^p, V^p and b^p are independent. Line 8-14 update corresponding rows of matrices U^p, V^p, b^p based on the sampled instance (s_p, o^+_p, o^-_p) in each iteration. As the sample generation step in line 4 is prior to the model parameter learning, thus the convergence criteria of Algorithm 1 is to iterate over all the sampled triples in D_p.

Given the constructed G_p as input, the time complexity of the update rules shown in Equations 4.9 4.10 4.11 4.12 4.13 is $\mathcal{O}(cK)$, where K is the number of latent features. The total computational complexity of Algorithm 1 is then $\mathcal{O}(|D_p| \cdot cK)$, where $|D_p|$ is the total size of pre-sampled triples shown in line 4 of Algorithm 1.

Algorithm 1 Bayesian Personalized Ranking Based Latent Feature Embedding Model

Input: latent dimension K, G, target predicate p
Output: U^p, V^p, b^p

1: Given target predicate p and entire knowledge graph G, construct its bipartite subgraph, G_p
2: m = number of subject entities in G_p
3: n = number of object entities in G_p
4: Generate a set of training samples $D_p = \{(s_p, o^+_p, o^-_p)\}$ using uniform sampling technique
5: Initialize U^p as size $m \times K$ matrix with 0 mean and standard deviation 0.1
6: Initialize V^p as size $n \times K$ matrix with 0 mean and stardard deviation 0.1
7: Initialize b^p as size $n \times 1$ column vector with 0 mean and stardard deviation 0.1
8: **for all** $(s_p, o^+_p, o^-_p) \in D_p$ **do**
9: Update U^p_s based on Equation 4.9
10: Update $V^p_{o^+}$ based on Equation 4.10
11: Update $V^p_{o^-}$ based on Equation 4.11
12: Update $b^p_{o^+}$ based on Equation 4.12
13: Update $b^p_{o^-}$ based on Equation 4.13
14: **end for**
15: **return** U^p, V^p, b^p

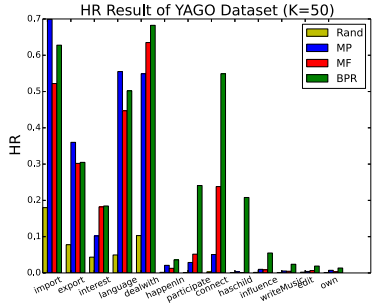
5 Experiments and Results

This section presents our experimental analysis of the Algorithm 1 for thirteen unique predicates in the well known YAGO2 knowledge graph [12]. We construct a model for each predicate and describe our evaluation strategies, including performance metrics and selection of state-of-the-art methods for benchmarking in section 5.1. We aim to answer two questions through our experiments:

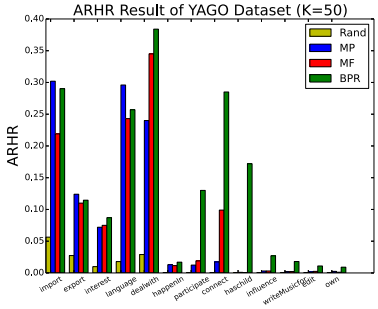
1. How does our approach compare with related work for link recommendation in knowledge graph?

2. For a predicate p, can we reason about the link prediction model performance M_p in terms of the structural metrics of the bipartite graph G_p?

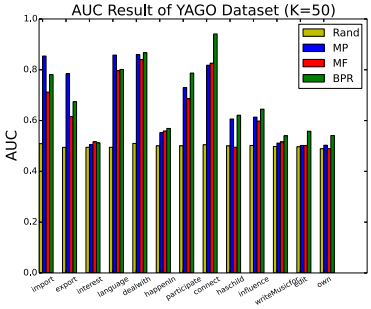
Table 1 shows the statistic of various YAGO2 relations used in our experiments. # Subjects and # Objects represent the number of subject and object entities associated with its corresponding predicate. The last column shown in Table 1 shows the number of facts for each relation in YAGO2. We run all the experiments on a 2.1 GHz Machine with 4GB memory running Linux operating system. The algorithms are implemented in Python language along with NumPy and SciPy libraries for linear algebra operations. The



(a) HR Comparison among different link recommendation methods

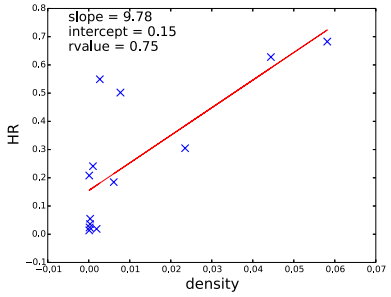


(b) ARHR Comparison among different link recommendation methods

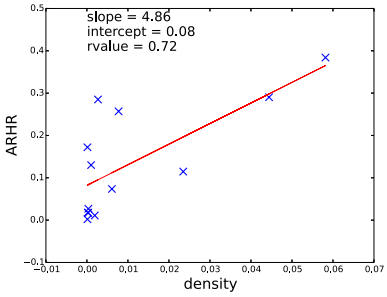


(c) AUC Comparison among different link recommendation methods

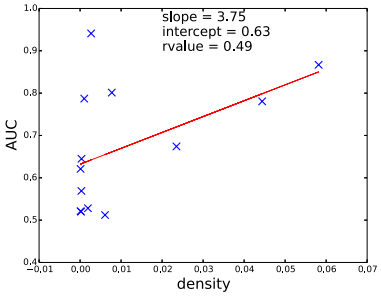
Figure 1: Link Recommendation Comparison on YAGO2 Relations



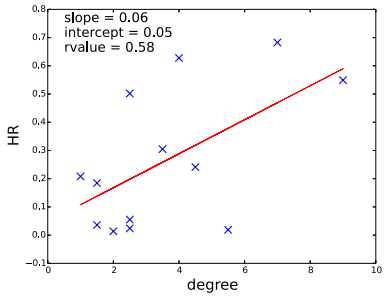
(a) Graph Density and HR



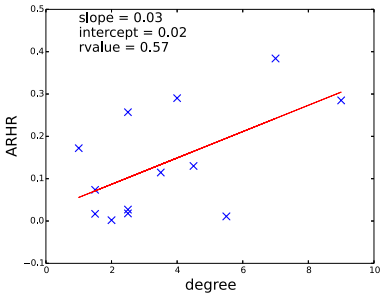
(b) Graph Density and ARHR



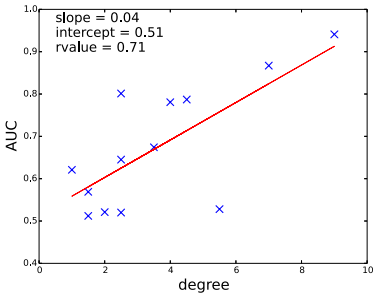
(c) Graph Density and AUC



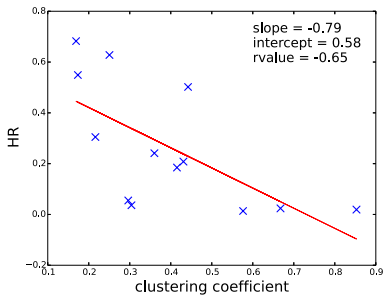
(d) Graph Average Degree and HR



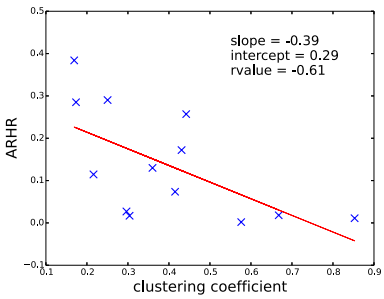
(e) Graph Average Degree and ARHR



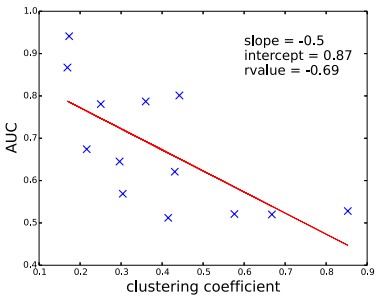
(f) Graph Average Degree and AUC



(g) Clustering Coefficient and HR



(h) Clustering Coefficient and ARHR



(i) Clustering Coefficient and AUC

Figure 2: Quantitative Analysis Between Graph Topology and Link Recommendation Model Performance

Relation	# Subjects	# Objects	# of Facts in YAGO2
Import	142	62	391
Export	140	176	579
isInterestedIn	358	213	464
hasOfficialLanguage	583	214	964
dealsWith	131	124	945
happenedIn	7121	5526	12500
participatedIn	2330	7043	16809
isConnectedTo	2835	4391	33581
hasChild	10758	12800	17320
influence	8056	9153	25819
wroteMusicFor	5109	21487	24271
edited	549	5673	5946
owns	8330	24422	26536

Table 1: Statistics of Various Relations in YAGO2 Dataset

software is available online for download [2].

5.1 Experimental Setting

For our experiment, in order to demonstrate the performance of our proposed link prediction model, we use the YAGO2 dataset and several evaluation metrics for all compared algorithms. Particularly, for each relation, we split the data into a training part, used for model training, and a test part, used for model evaluation. We apply 5-time leave one out evaluation strategy, where for each subject, we randomly remove one fact (one subject-object pair) and place it into test set S_{test} and remaining in the training set S_{train}. For every subject, the training model will generate a size-N ranked list of recommended objects for recommendation task. The evaluation is conducted by comparing the recommendation list of each subject and the object entity of that subject in the test set. Grid search is applied to find regularization parameters, and we set the values of parameters used in section 4.2 as $\lambda_s = \lambda_{o^+} = \lambda_{o^-} = 0.005$. For other model parameters, we fix learning rate $\alpha = 0.2$, and number of latent factors $K = 50$ respectively. For parameter in model evaluation, we set $N = 10$.

In order to illustrate the merit of our proposed approach, we compare our model with the following methods for link prediction in a knowledge graph. Since the problem we solve in this paper is similar to the one-class item recommendation [23] in recommender system domain, we consider the following state-of-the-art one-class recommendation methods as baseline approaches for comparison.

1. **Random (Rand):** For each relation, this method randomly selects subject-object entity pair for link recommendation task.
2. **Most Popular (MP):** For each predicate in knowledge base, this method presents a non-personalized ranked object list based on how often object entities are connected among all subject entities.
3. **MF:** The matrix factorization method is proposed by [16], which uses a point-wise strategy for solving the one-class item recommendation problem.

During the model evaluation stage, we use three popular metrics, namely Hit Rate (HR), Average Reciprocal Hit-Rank (ARHR), and Area Under Curve (AUC), to measure the link recommendation quality of our proposed approach in comparison to baseline methods. HR is defined as follows:

$$HR = \frac{\#hits}{\#subjects} \tag{5.14}$$

where $\#subjects$ is the total number of subject entities in test set, and $\#hits$ is the number of subjects whose object entity in the test set is recommended in the size-N recommendation list. The second evaluation metric, ARHR, considering the ranking of the recommended object for each subject entity in knowledge graph, is defined as below:

$$ARHR = \frac{1}{\#subjects} \sum_{i=1}^{\#hits} \frac{1}{p_i} \tag{5.15}$$

where if an object of a subject is recommended for connection in knowledge graph which we name as hit under this scenario, p_i is the position of the object in the ranked recommendation list. As we can see, ARHR is a weighted version of HR and it captures the importance of recommended object in the recommendation list.

The last metric, AUC is defined as follows:

$$AUC = \frac{1}{\#subjects} \sum_{s \in subjects} \frac{1}{|E(s)|} \sum_{(o^+,o^-) \in E(s)} \delta(x_{s,o^+} > x_{s,o^-}) \tag{5.16}$$

Where $E(s) = \{(o^+, o^-)|(s, o^+) \in S_{test} \cap (s, o^-) \notin (S_{test} \cup S_{train})\}$, and $\delta()$ is the indicator function.

For all of three metrics, higher values indicate better model performance. Specifically, the trivial AUC of a random predictor is 0.5 and the best value of AUC is 1.

5.2 YAGO2 Relation Prediction Performance

Figure 1 shows the average link prediction performance for YAGO2 relations using various methods. Our proposed latent feature embedding approach

[2] https://sites.google.com/site/baichuanzhangpurdue/sample-code-and-dataset

shows overall improvement compared with other algorithms on most of relations in YAGO2. For instance, for all the YAGO2 predicates used in the experiment, our proposed model consistently outperforms MF based method, which demonstrates the empirical experience that pairwise ranking based method achieves much better performance than pointwise regression based method given implicit feedback for link recommendation task. Compared with Popularity based recommendation method MP, our method obtains better performance for most predicates. For example, predicates such as "participate","connect","hasChild", and "influence", our proposed model achieves more than 10 times better performance in terms of both HR and ARHR. However, for several predicates such as "import", "export", and "language", MP based method performs the best among all the competing methods. The good performance of MP is owing to the semantic meaning of specific predicate. For instance, "import" represents Country/Product relation in YAGO2, which indicates the types of its subject and object entities are geographic region and commodity respectively. For such a predicate, most popular object entities such as food, cloth, fuel are linked to most of the countries, which helps MP based method obtain good link recommendation performance.

5.3 Analysis and Discussion

Figure 1 shows that the link prediction model performance widely varies from predicate to predicate in the YAGO2 knowledge base. For example, the HR of predicate "dealsWith" is significantly better than "own". Thus it is critical that we quantitatively understand the model performance across various relations in a knowledge graph. Recall from the **Problem Statement** that given a predicate p, our model M_p only accounts for the bipartite subgraph G_p. Motivated by [19], we study the impact of resultant graph structure of G_p on the performance of M_p.

For each predicate p, we compute several graph topology metrics on its bipartite subgraph G_p such as graph density, graph average degree, and clustering coefficient. Figure 2 shows the quantitative analysis between graph structure and link prediction model performance of each predicate. In each subfigure, x-axis represents the computed graph topology metric value of each predicate and y-axis denotes our proposed link prediction model performance in terms of HR, ARHR, and AUC. Each cross point shown in blue represents one specific YAGO2 predicate used in our experiments. Then we developed a linear regression model to understand the correlation between link prediction model performance and each graph metric. For each linear regression curve shown in red color, we also report its slope, intercept, and correlation coefficient (rvalue) to capture the association trend.

From Figure 2, both graph density and graph average degree show strong positive correlation signal with proposed link prediction model as demonstrated by rvalue. As our approach is inspired by collaborative filtering for recommender systems that accept a user-item matrix as input, for resultant graph of each predicate, higher graph density indicates higher matrix density in user-item matrix, which naturally leads to better recommendation performance in recommender system domain. Similar explanation can be adapted to graph average degree. For the clustering coefficient, it shows strong negative correlation signal with link prediction model performance. For instance, in terms of AUC, the rvalue is around -0.69. As clustering coefficient (cc) is the number of closed triples over the total number of triples in graph, smaller value of cc indicates lower fraction of closed triples in the graph. Based on the transitivity property of a social graph, which states the friends of your friend have high likelihood to be friends themselves [26, 30], it is relatively easier for link prediction model to predict (i.e.,hit) such link with open triple property in the graph, which leads to better link prediction performance.

6 Conclusion and Future Work

Inspired by the success of collaborative filtering algorithms for recommender systems, we propose a latent feature based embedding model for the task of link prediction in a knowledge graph. Our proposed method provides a measure of "confidence" for adding a triple into the knowledge graph. We evaluate our implementation on the well known YAGO2 knowledge graph. The experiments show that our Bayesian Personalized Ranking based latent feature embedding approach achieves better performance compared with two state-of-art recommender system models: Most Popular and Matrix Factorization. We also develop a linear regression model to quantitatively study the correlation between the performance of link prediction model itself and various topological metrics of the graph from which the models are constructed. The regression analysis shows strong correlation between the link prediction performance and graph topological features, such as graph density, average degree and clustering coefficient.

For a given predicate, we build link prediction models solely based on the bipartite subgraph of the original knowledge graph. However, as real-world experience suggests, the existence of a relation between two entities can also be predicted from the presence of other rela-

tions, either direct or through common neighbors. As an example, the knowledge of where someone studies and who they are friends with is useful to predict possible workplaces. Incorporating such intuition as "social signals" into our current model will be the prime candidate for an immediate future work. Another future work would be to update the knowledge graph based on the newer facts that become available over time in streaming data sources.

Acknowledgement

This work was supported by the Analysis In Motion Initiative at Pacific Northwest National Laboratory, which is operated by Battelle Memorial Institute, and by Mohammad Al Hasan's NSF CAREER Award (IIS-1149851).

References

[1] A. Bordes and E. Gabrilovich. Constructing and mining web-scale knowledge graphs. In *SIGKDD 14*.

[2] R. Bro. Parafac. tutorial and applications. *Chemometrics and Intelligent Laboratory Systems*, 1997.

[3] K.-W. Chang, W. tau Yih, B. Yang, and C. Meek. Typed tensor decomposition of knowledge bases for relation extraction. ACL, 2014.

[4] P. Chen and A. O. H. III. Deep community detection. *IEEE Transactions on Signal Processing*, 2015.

[5] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. Natural language processing (almost) from scratch. *JMLR*, 2011.

[6] X. Dong, E. Gabrilovich, G. Heitz, W. Horn, N. Lao, K. Murphy, T. Strohmann, S. Sun, and W. Zhang. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *SIGKDD*, 2014.

[7] L. Drumond, S. Rendle, and L. Schmidt-Thieme. Predicting rdf triples in incomplete knowledge bases with tensor factorization. In *ACM SAC*, 2012.

[8] O. Etzioni, M. Banko, S. Soderland, and D. S. Weld. Open information extraction from the web. *Communications of the ACM*, pages 68–74, 2008.

[9] D. A. Ferrucci, E. W. Brown, J. Chu-Carroll, J. Fan, D. Gondek, A. Kalyanpur, A. Lally, J. W. Murdock, E. Nyberg, J. M. Prager, N. Schlaefer, and C. A. Welty. Building watson: An overview of the deepqa project. *AI Magazine*, 2010.

[10] N. Friedman, L. Getoor, D. Koller, and A. Pfeffer. Learning probabilistic relational models. In *IJCAI 99*.

[11] M. A. Hasan, V. Chaoji, S. Salem, and M. Zaki. Link prediction using supervised learning. In *In Proc. of SDM 06 workshop on Link Analysis, Counterterrorism and Security*, 2006.

[12] J. Hoffart, F. M. Suchanek, K. Berberich, E. Lewis-Kelham, G. de Melo, and G. Weikum. Yago2: Exploring and querying world knowledge in time, space, context, and many languages. In *WWW*, 2011.

[13] R. Jenatton, N. L. Roux, A. Bordes, and G. R. Obozinski. A latent factor model for highly multi-relational data. In *NIPS*. 2012.

[14] S. Jiang, D. Lowd, and D. Dou. Learning to refine an automatically extracted knowledge base using markov logic. In *ICDM*, pages 912–917, 2012.

[15] X. Jiang, V. Tresp, Y. Huang, and M. Nickel. Link prediction in multi-relational graphs using additive models. In *SeRSy*, pages 1–12, 2012.

[16] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Journal of Computer Science*, pages 30–37, 2009.

[17] N. Lao and W. W. Cohen. Relational retrieval using a combination of path-constrained random walks. *Journal of Machine learning*, pages 53–67, 2010.

[18] N. Lao, T. Mitchell, and W. W. Cohen. Random walk inference and learning in a large scale knowledge base. In *EMNLP*, pages 529–539, 2011.

[19] D. Liben-Nowell and J. Kleinberg. The link prediction problem for social networks. CIKM, 2003.

[20] M. Nickel, X. Jiang, and V. Tresp. Reducing the rank in relational factorization models by including observable patterns. In *NIPS*, pages 1179–1187, 2014.

[21] M. Nickel, K. Murphy, V. Tresp, and E. Gabrilovich. A review of relational machine learning for knowledge graphs: From multi-relational link prediction to automated knowledge graph construction. *arXiv preprint arXiv:1503.00759*, 2015.

[22] M. Nickel, V. Tresp, and H. peter Kriegel. A three-way model for collective learning on multi-relational data. In *ICML*, pages 809–816, 2011.

[23] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In *UAI*, 2009.

[24] M. Richardson and P. Domingos. Markov logic networks. *Machine learning*, pages 107–136, 2006.

[25] S. Riedel, L. Yao, A. McCallum, and B. M. Marlin. Relation extraction with matrix factorization and universal schemas. In *HLT-NAACL*, 2013.

[26] T. K. Saha, B. Zhang, and M. Al Hasan. Name disambiguation from link data in a collaboration graph using temporal and topological features. *Social Network Analysis and Mining*, pages 1–14, 2015.

[27] R. Socher, D. Chen, C. D. Manning, and A. Ng. Reasoning with neural tensor networks for knowledge base completion. In *NIPS*, pages 926–934, 2013.

[28] C. H. Tan, E. Agichtein, P. Ipeirotis, and E. Gabrilovich. Trust, but verify: Predicting contribution quality for knowledge base construction and curation. In *WSDM*, pages 553–562, 2014.

[29] B. Taskar, P. Abbeel, and D. Koller. Discriminative probabilistic models for relational data. In *UAI*, 2002.

[30] B. Zhang, T. K. Saha, and M. Al Hasan. Name disambiguation from link data in a collaboration graph. In *ASONAM*, 2014.