Mining Marked Nodes in Large Graphs

by

Scott Freitas

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved April 2018 by the
Graduate Supervisory Committee:

Hanghang Tong, Chair
Ross Maciejewski
Yezhou Yang

ARIZONA STATE UNIVERSITY

May 2018

ABSTRACT

With the rise of the Big Data Era, an exponential amount of network data is being generated at an unprecedented rate across a wide-range of high impact micro and macro areas of research—from protein interaction to social networks. The critical challenge is translating this large scale network data into actionable information.

A key task in the data translation is the analysis of network connectivity via marked nodes—the primary focus of our research. We have developed a framework for analyzing network connectivity via marked nodes in large scale graphs, utilizing novel algorithms in three interrelated areas: (1) analysis of a single seed node via its ego-centric network (AttriPart algorithm); (2) pathway identification between two seed nodes (K-Simple Shortest Paths Multithreaded and Search Reduced (KSSPR) algorithm); and (3) tree detection, defining the interaction between three or more seed nodes (Shortest Path MST algorithm).

In an effort to address both fundamental and applied research issues, we have developed the LocalForcasting algorithm to explore how network connectivity analysis can be applied to local community evolution and recommender systems. The goal is to apply the LocalForecasting algorithm to various domains—e.g., friend suggestions in social networks or future collaboration in co-authorship networks. This algorithm utilizes link prediction in combination with the AttriPart algorithm to predict future connections in local graph partitions.

Results show that our proposed AttriPart algorithm finds up to 1.6× denser local partitions, while running approximately 43× faster than traditional local partitioning techniques (PageRank-Nibble). In addition, our LocalForecasting algorithm demonstrates a significant improvement in the number of nodes and edges correctly predicted over baseline methods. Furthermore, results for the KSSPR algorithm demonstrate a speed-up of up to 2.5× the standard k-simple shortest paths algorithm.

ACKNOWLEDGMENTS

I'd like to thank Dr. Tong for all his mentorship over the past two years. Without his guidance, this research would not have been possible. I would also like to thank Dr. Maciejewski and Dr. Yang for all the time that they spent helping me navigate the unknowns of graduate school. Finally, I'd like to thank my dad, Gary Freitas, for all of the time he has spent editing and listening to my research—his support and advice have been invaluable.

TABLE OF CONTENTS

Page

CHAPTER Page

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

Chapter 1

INTRODUCTION

Every day, enormous amounts of network data is being generated in a variety of fields ranging from social media to health. This research is an attempt to analyze this network data from the viewpoint of seed nodes in a graph—with the goal of understanding how these seed nodes relate to one another, and to their surrounding neighbors. However, real world networks often subject to two important factors, (1) they are typically rich graphs with attribute data and (2) they can be incredibly large—up to hundreds of millions of nodes and edges. This research will make use of this information and address it in the development of it's analysis and algorithmic formulations.

1.1 Acknowledgments

This work is partially supported by the National Science Foundation under Grant No. IIS-1651203, IIS-1715385 and IIS-1743040, by DTRA under the grant number HDTRA1-16-0017, by Army Research Office under the contract number W911NF-16-1-0168, and gifts from Huawei and Baidu.

1.2 Objective

The fundamental research objectives are three fold—(1) analyze the network connectivity of a graph given a set of two or more seed nodes, (2) determine the ego-centric graph of a given seed node by means of local graph partitioning and (3) make the outcomes of this research accessible and interactive to the user. To accomplish this we propose a series of sub-goals.

(1) To determine the network connectivity between two or more seed nodes we plan to use two algorithms—(a) k-simple shortest paths and (b) minimum spanning tree for a subset of nodes. In addition to the baseline versions of these algorithms we propose to introduce novel methods of speeding up the network analysis computations by using (i) parallel processing, (ii) network search space reduction and (iii) subsequent graph re-query of prior network analysis.

(2) To analyze the ego-centric graph of a given seed node by means of local graph partitioning we introduce AttriPart which incorporates the additional information found in rich graphs instead of using the network structure (Nibble Spielman and Teng (2013) and PageRank-Nibble Andersen *et al.* (2006)).

1.3 Importance

This research has broad applications in a wide variety of use cases. Three of these major use cases are (1) social media, (2) map routing and (3) internet connectivity.

In terms of (1) social media analysis, two important foundational research questions this thesis addresses are—(a) how do you quickly determine how a given user relates to one ore more people in a network and (b) how do you find better ego-centric graphs by incorporating attribute information as opposed to using topological connections alone.

Once these foundational research questions are addressed, we can ask more applicative questions such as—(a) how does a local community evolve over time, (b) how can you create enhanced user or product recommendations, (c) what kind of knowledge can be extracted from the network connectivity and local graph partitioning and (d) what kind of decisions can be made based on this extracted knowledge.

When looking at (2) map routing networks, this research can allow us to quickly find the k-simple shortest paths between the place you're currently at and where you want to go (e.g. destination routing). An example where the ‘k’ part of the shortest paths can be useful is when accidents exist on the shortest route and secondary routes need to be taken. In addition, we might need to find the most important landmarks or businesses in the area relative to your current position. This could be done using local graph partitioning algorithms.

When analyzing (3) internet connectivity between servers, computers, routers, etc., it can be important to find the k-simple shortest paths between a source and destination. This kind of analysis is useful to identify the flow of information in the internet and what routes are available for data transmission.

1.4 Completion Criteria

The criteria to complete the thesis will include three parts. (1) The implementation of the network connectivity algorithms, including the search space reduction algorithm, (2) the implementation of the local graph partitioning algorithm that can utilize the information in rich graph attributes for enhanced analysis and (3) the development of a web platform (PathFinder: `www.path-finder.io`) to visualize the network connectivity.

Chapter 2

LITERATURE SURVEY

2.1 Local Graph Partitioning

The goal of local clustering algorithms is to find a set of vertices in the surrounding area of a given seed node such that the set of vertices forms a tight-knit group defined by the conductance of the set. Converse to the problem of local clustering, global clustering algorithms take into account the whole network in order to identify all of the existing network clusters. Both local and global clustering are areas of significant research and as such we will provide a short, high-level overview of a select few topics with a focus on the research that pertains to the algorithms we present.

We provide a high level review of both local and global community detection methods in addition to random walks, with a focus on the research that pertains to the algorithms we propose in this paper.

A - Local Community Detection. Given an undirected graph, start vertex and a target conductance—the goal of Nibble is to find a subset of vertices that has conductance less than the target conductance Spielman and Teng (2013). This algorithm has strong theoretical properties with a run time of $O(2^b(log^6 m)/\phi^4)$, where b is a user defined constant, ϕ is the target conductance and m is the number of edges. PageRank-Nibble builds on the work of Nibble by introducing the use of personalized PageRank Haveliwala (2003); Tong *et al.* (2006), in addition to an algorithm for the computation of approximate PageRank vectors Andersen *et al.* (2006). Since PageRank-Nibble and Nibble run on undirected graphs, they use truncated random walks in order to prevent the stationary distribution from becoming proportional to the degree centrality of each node Grolmusz (2015). There are also many alternative techniques for local community detection. To name a few, the paper by Bagrow and Bollt Bagrow and Bollt (2005) introduces a method of local community identification that utilizes an l-shell spreading outward from a start vertex. However, their algorithm requires knowledge of the entire graph and is therefore not truly local. The research by J. Chen et. al. Chen *et al.* (2009) proposes a method for local community identification in social networks that avoids the use of hard to obtain parameters and improves the accuracy of identified communities by introducing a new metric. In addition, the work by Zhou *et al.* (2017) and Yin *et al.* (2017) introduces two methods of local community identification that take into account high-order network structure information. In Zhou *et al.* (2017), the authors provide mathematical guarantees of the optimality and scalability of their algorithms, in addition to the generalization of it to various network types (e.g. signed and multi-partite networks).

B - Global Community Detection. The basic idea behind the Walktrap algorithm is that random walks on a graph tend to get "trapped" in densely connected parts that correspond to communities Pons and Latapy (2005). Utilizing the properties of random walks on graphs, they define a measurement of structural similarity between vertices and between communities, creating a distance metric. The algorithm itself has an upper bound of $O(mn^2)$. Another popular choice for global community detection is spectral analysis. In the paper by M. Newman Newman (2013) it is shown that the problems of community detection by modularity maximization, community detection by statistical inference and normalized-cut graph partitioning when tackled using spectral methods, are in fact, the same problem. The work by S. White et. al. in White and Smyth (2005) attempts to find communities in graphs using spectral clustering. They achieve this by using an objective function for graph clustering Newman and Girvan (2004) and reformulating it as a spectral relaxation problem, for which they propose two algorithms to solve it. A systematic introduction to spectral clustering techniques can be found in von Luxburg (2007). There also exists many alternative techniques for global community detection. Among others, two interesting techniques relevant to this work are Jaewon yang (2013) Takaffoli *et al.* (2014). In Jaewon yang (2013), the authors propose a community detection algorithm that uses the information in both the network structure and the node attributes, while in Takaffoli *et al.* (2014) the authors use network feature extraction to predict the evolution of communities. A detailed review of various community detection algorithms can be found in Zhao Yang (2016).

C - Random Walks. Random walks have been utilized in many forms, two of which we focus on in this paper. (1) As a form of node ranking in graphs with use of PageRank Page *et al.* (1998) and it's extension to personalized PageRank Haveliwala (2003). (2) As a method of subgraph identification Dupont *et al.* (2017) around a given seed node. In Dupont *et al.* (2017) the authors attempt to determine the relevant subgraph between two or mode seed nodes in a graph using the technique of random betweenness centrality introduced by M. Newman Newman (2005).

2.2 Network Connectivity

There has been a significant amount of research related to determining the network connectivity between a set of seed nodes in a graph. In addition, the concept of user-specific query nodes (seed nodes) has been an active research topic. For example, Staudt et. al. used user-specific query nodes for community detection via “selSCAN” Staudt *et al.* (2014) and Akoglu et. al. used them to find connection pathways Akoglu *et al.* (2013). Below, we discuss (1) the related work to the two core algorithms we use to analyze the network connectivity—K-Simple Shortest Paths and Shortest Paths MST; (2) related work to the search space reduction algorithm—Key Neighboring

Vertices—utilized by the network connectivity algorithms to reduce the run time of connectivity analysis.

A - Two Seed Nodes: K-Simple Shortest Paths. A theoretical analysis has been laid out by Ruppert Ruppert (2012), along with the theoretical and experimental work by Guerriero et. al using a shared memory model Guerriero and Musmanno (2000). More recently, Singh et. al. used GPUs along with CUDA to parallelize the algorithm, resulting in impressive speedup Singh and Singh (2015). In addition, two representative works for connection subgraph identification (i.e. pathway detection) can be seen in Akoglu *et al.* (2013) and Faloutsos *et al.* (2004).

B - Two Seed Nodes: Shortest Paths MST. Related work towards creating a minimum spanning tree for a susbset of nodes has been done by Cenek et. al. Instead of creating a shortest paths distance matrix, they proposed to insert the shortest paths into the tree dynamically Cenek and Hrcka (2015).

C - Search Space Reduction: Key Neighboring Vertices. Sulieman et. al. proposed a semantic social breadth first search algorithm which takes the top N vertices with the highest centrality degree and attempts to find the nearby influential players D. Sulieman (2013).

Chapter 3

LOCAL PARTITIONING IN RICH GRAPHS

Local graph partitioning is a key graph mining tool that allows researchers to identify small groups of interrelated nodes (e.g. people) and their connective edges (e.g. interactions). Because local graph partitioning is primarily focused on the network structure of the graph (vertices and edges), it often fails to consider the additional information contained in the attributes. In this paper we propose—(i) a scalable algorithm to improve local graph partitioning by taking into account both the network structure of the graph and the attribute data and (ii) an application of the proposed local graph partitioning algorithm (AttriPart) to predict the evolution of local communities (LocalForecasting). Experimental results show that our proposed AttriPart algorithm finds up to 1.6× denser local partitions, while running approximately 43× faster than traditional local partitioning techniques (PageRank-Nibble Andersen *et al.* (2006)). In addition, our LocalForecasting algorithm shows a significant improvement in the number of nodes and edges correctly predicted over baseline methods.

3.1 Introduction

Motivation. With the rise of the big data era an exponential amount of network data is being generated at an unprecedented rate across many disciplines. One of the critical challenges before us is the translation of this large-scale network data into meaningful information. A key task in this translation is the identification of local communities with respect to a given seed node [1] . In practical terms, the

[1] we interchangeably refer to local community as a local partition

information discovered in these local communities can be utilized in a wide range of high-impact areas—from the micro (protein interaction networks Laura Bennett (2014) Yong-Yeol Ahn (2010)) to the macro (social Tantipathananandh *et al.* (2007) Chen *et al.* (2009) and transportation networks).

Problem Overview. How can we quickly determine the local graph partition around a given seed node? This problem is traditionally solved using an algorithm like Nibble Spielman and Teng (2013), which identifies a small cluster in time proportional to the size of the cluster, or PageRank-Nibble, Andersen *et al.* (2006) which improves the running time and approximation ratio of Nibble with a smaller polylog time complexity. While both of these methods provide powerful techniques in the analysis of network structure, they fail to take into account the attribute information contained in many real-world graphs. Other techniques to find improved rank vectors, such as attributed PageRank Hsu *et al.* (2017), lack a generalized conductance metric for measuring cluster ”goodness” containing attribute information. In this paper, we propose a novel method that combines the network structure and attribute information contained in graphs—to better identify local partitions using a generalized conductance metric.

Applications. Local graph partition plays a central role in many application scenarios. For example, a common problem in recommender systems is that of social media networks and determining how a local community will evolve over time. The proposed LocalForecasting algorithm can be used to determine the evolution of local communities, which can then assist in user recommendations. Another example utilizing social media networks is ego-centric network identification, where the goal is to identify the locally important neighbors relative to a given person. To this end, we can use our AttriPart algorithm to identify better ego-centric networks using the graph's network structure and attribute information. Finally, newly arrived nodes (i.e., cold-start nodes) often contain few connections to their surrounding neighbors, making it difficult to ascertain their grouping to various communities. The proposed LocalForecasting algorithm mitigates this problem by introducing additional attribute edges (link prediction), which can assist in determining which local partitions the cold start nodes will belong to in the future.

Contributions. Our primary contributions are three-fold:

- The formulation of a graph model and generalized conductance metric that incorporates both attribute and network structure edges.
- The design and analysis of local clustering algorithm AttriPart and local community prediction algorithm LocalForecasting. Both algorithms utilize the proposed graph model, modified conductance metric and novel subgraph identification technique.
- The evaluation of the proposed algorithms on three real-world datasets—demonstrating the ability to rapidly identify denser local partitions compared to traditional techniques.

Deployment. The local partitioning algorithm AttriPart is currently deployed to the PathFinder Freitas *et al.* (2017) web platform (`www.path-finder.io`), with the goal of assisting users in mining local network connectivity from large networks. The design and deployment challenges were wide ranging, including—(i) the integration of four different programming languages, (ii) obtaining real-time performance with low cost hardware and (iii) implementation of a visually appealing and easy to use interface. We note that the AttriPart algorithm, deployed to the web platform, has performance nearly identical to the results presented in section 3.4.

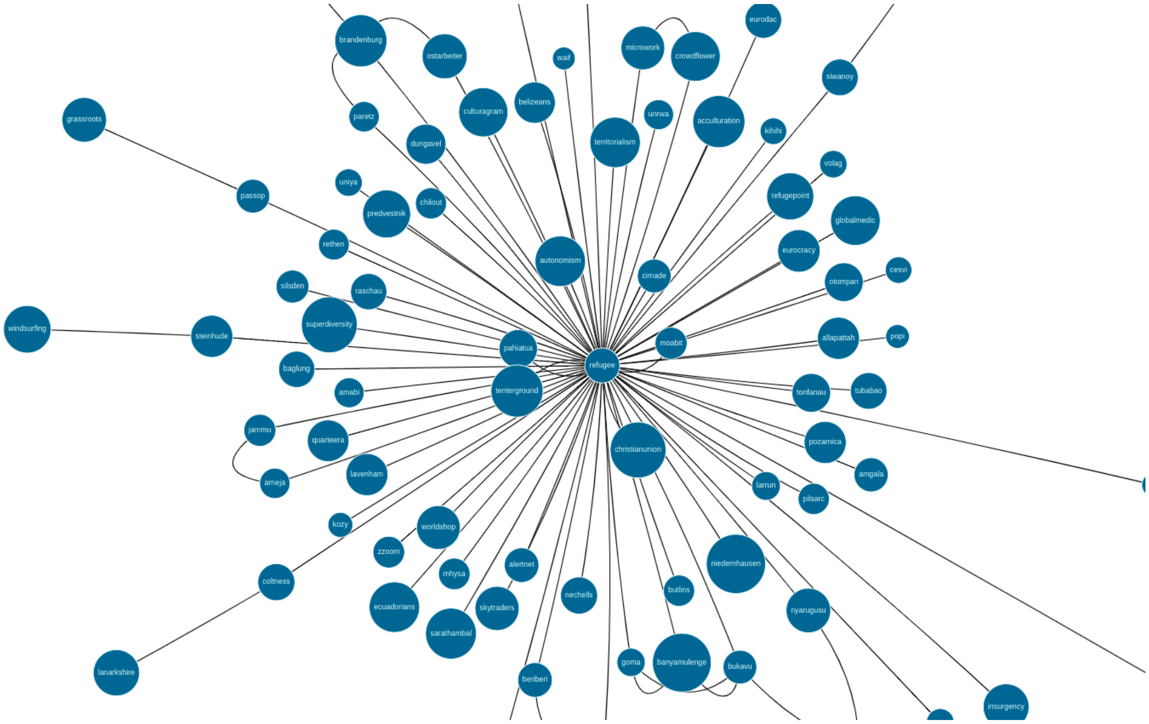


Figure 3.1: Close-up of the AttriPart algorithm on the PathFinder web platform.

This paper is organized as follows—Section 2 defines the problem of local partitioning in rich graphs; Section 3 introduces our proposed model and algorithms; Section 4 presents our experimental results on multiple real-world datasets; Section 5 reviews the related literature; and Section 6 concludes the paper.

3.2 Problem Definition

In this paper we consider three graphs—(1) an undirected, unweighted structure graph $G = (V, E)$, (2) an undirected, weighted attribute graph $A = (V, E)$ and (3) a combined graph consisting of both G and A that is undirected and weighted $B = (V, E)$. In each graph, V is the set of vertices, E is the set of edges, n is the number of vertices and m is the number of edges (i.e. G, H and B contain the same number of vertices and edges by default). In order to denote the degree centrality we say $\delta(v)$ is the degree of vertex v. We use uppercase letters to denote matrices (e.g. G) and lowercase letters to denote vectors (e.g. v).

For the ease of description, we define terms that are interchangeably used throughout the literature and this paper—(a) we refer to network as a graph, (b) node is synonymous with vertex, (c) local partition is referred to as a local cluster, (d) seed node is equivalent to query and start vertex, (e) topological edges of the graph refers to the network structure of the graph, (f) a rich graph is a graph with attributes on the nodes and or edges.

Having outlined the notation, we define the problem of local partitioning in rich graphs as follows:

Problem 1. Local Partitioning in Rich Graphs

Given: (1) an undirected, unweighted graph $G = (V, E)$, (2) a seed node $q \in V$ and (3) attribute information for each node $v \in V$ containing a k-dimensional attribute vector x_i—with an attribute matrix $X = [x_1, x_2, ..., x_n] \in \mathbb{R}^{k \times n}$ representing the attribute vector for each node v.

Output: a subset of vertices $S \subset V$ such that S best represents the local partition around seed node q in graph B.

Table 3.1: Symbols and Definition

Symbol	Definition
G, A, B	network, attribute & combined graphs
n, m	number of nodes & edges in graphs G, A, B
m_e	number of edges in B after LocalForecasting
p, m_p	number of nodes & edges in T
s, q, ϕ_o	preference vector, seed node & target conductance
W	lazy random walk transition matrix
S	set of vertices representing local partition
ϵ, ϵ_t	rank truncation and iteration thresholds
t_m, n_s	rank vector iterations; number of vertices to sweep
α_n, α_r	AttriPart & LocalProximity teleport values
t_s, n_w	subgraph relevance threshold & number of walks
T; D, L	subgraph of B; walk count dictionary & list
$\mu(L)$, $\sigma(L)$	mean and standard deviation of L
t_e	edge addition threshold

3.3 Methodology

This section first describes the preliminaries for our proposed algorithms, including the graph model and modified conductance metric. Next, we introduce each proposed algorithm—(1) LocalProximity, (2) AttriPart and (3) LocalForecasting. Finally, we provide an analysis of the proposed algorithms in terms of effectiveness and efficiency.

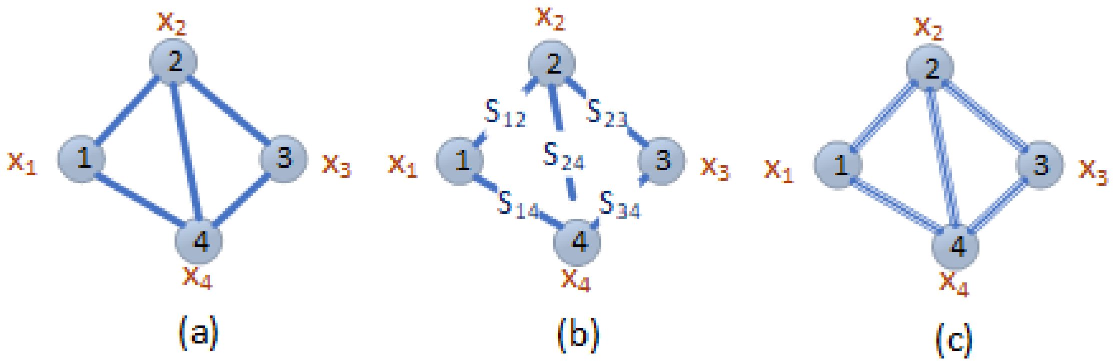


Figure 3.2:

Example of the three graph models: (a) graph G is the network structure with nodes $\{1, 2, 3, 4\}$ and corresponding attribute set $\{x_1, x_2, x_3, x_4\}$ given as input. (b) Graph A is the attribute network with the same set of edges as G with each edge (u, v) assigned a positive similarity weight s_{uv}. (c) Graph B is a linear combination of each respective edge (u, v) from G and A.

3.3.1 Preliminaries

Graph Model. Topological network G represents the network structure of the graph and is formally defined in Eq. (3.1). Attribute network A represents the attribute structure of the graph and is computed based on the similarity for every edge $(u, v) \in E$ in G. In order to determine the similarity between the two nodes, we use Jaccard Similarity $J(u, v)$. A is formally defined in Eq. (3.2) where 0.05 is the default attribute similarity between an edge $(u, v) \in E$ in G if $J(x_u, x_v) = 0$. In addition, t_e is the similarity threshold for the addition of edges not in G where $0 < t_e \leq 1$. Combined Network B represents the combined graph of G and A and is formally defined in Eq. (3.3).

Formally, we define each of the three graph models G, A and B in Eq. (3.1), Eq. (3.2) and Eq. (3.3). Figure 3.2 presents an illustrative example.

$$G(u,v) = \begin{cases} 1, & \text{if } (u,v) \in E \text{ and u}\neq\text{v} \\ 0, & \text{otherwise} \end{cases} \tag{3.1}$$

$$A(u,v) = \begin{cases} \text{J(u,v)}, & \text{if } (u,v) \in E,\ \text{u}\neq\text{v and } J(u,v) > 0 \\ 0.05, & \text{if } (u,v) \in E,\ \text{u}\neq\text{v and } J(u,v) = 0 \\ \text{J(u,v)}, & \text{if } (u,v) \notin E,\ \text{u}\neq\text{v and } J(u,v) > t_e \\ 0, & \text{otherwise} \end{cases} \tag{3.2}$$

$$B(u,v) = \begin{cases} \text{1+A(u,v)}, & \text{if } (u,v) \in E \text{ and } (u,v) \in A \\ \text{A(u,v)}, & \text{if } (u,v) \notin E \text{ and } (u,v) \in A \\ 0, & \text{otherwise} \end{cases} \tag{3.3}$$

Conductance. Conductance is a standard metric for determining how tight knit a set of vertices are in a graph Kannan *et al.* (2004). The traditional conductance metric is defined in Eq. (3.4), where S is the set of vertices representing the local partition. The lower the conductance value $\phi(S)$, where $0 \leq \phi(S) \leq 1$, the more likely S represents a good partition of the graph.

$$\phi(S) = \frac{cut(S)}{min(vol(S), vol(\bar{S}))} \tag{3.4}$$

Where the cut is $Cut(S) = \{(u,v) \in E | u \in S, v \notin S\}$, and the volume is $vol(S) = \sum\limits_{v \in S} \delta(v)$.

This definition of conductance will serve as the benchmark to compare the results of our parallel conductance metric.

Parallel Conductance. We propose a parallel conductance metric which takes into account both the attribute and topological edges in the graph. Instead of simply adding the cut of each vertex $v \in S$, we want to determine whether v is more similar to the vertices in S or $\bar{S}$. The new cut and conductance metric is formally defined in Eq. (3.5) and Eq. (3.6), respectively. The key idea behind the parallel conductance metric is to determine whether each vertex in S is more similar to S or $\bar{S}$ using the additional information provided by the attribute links.

$$parallel\_cut(S) = \sum_{i \epsilon S} \frac{\sum\limits_{j \not\epsilon S} B(i,j)}{\sum\limits_{j \epsilon S} B(i,j)} = \sum_{i \epsilon S} \frac{\sum\limits_{j \not\epsilon S} \left[A(i,j) + G(i,j)\right]}{\sum\limits_{j \epsilon S} \left[A(i,j) + G(i,j)\right]} \tag{3.5}$$

By definition, B can be split into its representative components, G and A. We also note a few key properties of the parallel cut metric below:

1. $Parallel\_cut = 1$ means that the vertices in S have connections of equal weighting between S and $\bar{S}$.

2. $Parallel\_cut < 1$ means that the vertices in S have only a few strong connections to $\bar{S}$.

3. $Parallel\_cut > 1$ means that the vertices in S are more strongly connected to $\bar{S}$ than S.

Eq. (3.6) uses the cut as defined in Eq. (3.5) and the volume as defined above with the modification that $\delta(v)$ is a sum of it's components in G and A.

$$\phi(S) = \frac{parallel\_cut(S)}{vol(S)} \tag{3.6}$$

We note that the parallel conductance metric has a different scale compared to the traditional conductance metric. For example, a conductance of 0.3 in the traditional conductance doesn't have the same meaning as a conductance of 0.3 in the parallel definition. We also bound the volume of S to $vol(S) < 1/2vol(B)$. This allows us to reduce the $min(vol(S), vol(\bar{S}))$ computation to $vol(S)$.

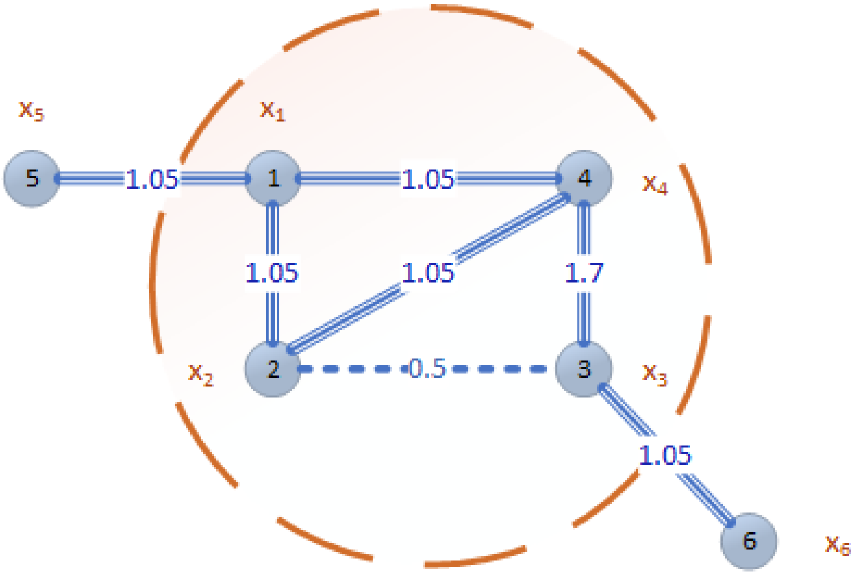


Figure 3.3: A toy example calculating the parallel cut and conductance with local partition S containing vertices $\{1, 2, 3, 4\}$. Parallel cut(V_1) = 1.05/2.1 = 0.5, parallel cut(V_2) = 0, parallel cut(V_3) = 1.05/2.2 = 0.477, parallel cut(V_4) = 0, parallel cut$(Total)$ = 0.5 + 0.477 = 0.977. Volume(S) = 12. Parallel conductance(S) = 0.977/12 = 0.0814.

3.3.2 Algorithms

We propose three algorithms in this subsection, including (1) LocalProximity (2) AttriPart and (3) LocalForecasting. First, we introduce the LocalProximity algorithm as a key building block for speeding-up the AttriPart and LocalForecasting algorithms by finding a subgraph containing only the nodes and edges relevant to the given seed node. Based on LocalProximity, we further propose the AttriPart algorithm to find a local partition around a seed node by minimizing the parallel conductance metric. Finally, we propose the LocalForecasting algorithm, which builds upon AttriPart, to predict a local community's evolution.

LocalProximity. There are two primary purposes for the LocalProximity algorithm—(i) the requisite computations for the LocalForecasting algorithm require a pairwise similarity calculation of all nodes, which is intractable for large graphs due to the quadratic run time. To make this computation feasible, we use the LocalProximity algorithm to determine a small subgraph of relevant vertices around a given seed node q. (ii) We experimentally found that the PageRank vector utilized in the AttriPart algorithm is significantly faster to compute after running the proposed LocalProximity algorithm.

Algorithm Details. The goal is to find a subgraph T around seed node q, such that T contains only nodes and edges likely to be reached in n_w trials of random walk with restart. We base the importance of a vertex $v \in V$ on the theory that random walks can measure the importance of nodes and edges in a graph Dupont *et al.* (2017)Newman (2005). This is done by defining node relevance proportional to the frequency of times a random walk with restart walks on a vertex in n_w trials (nodes walked on more than once in a walk will still count as one). Instead of using a simple threshold parameter to determine node/edge relevance as in Dupont *et al.* (2017), we utilize the mean and standard deviation of the walk distribution in order for the results to remain insensitive of n_w given that n_w is sufficiently large. In conjunction with the mean and standard deviation, we introduce t_s as a relevance threshold parameter to determine the size of the resulting subgraph T. See section 3.3.3 for more details.

Algorithm Description. The LocalProximity algorithm takes a graph B, a seed node $q \in B$, a teleport value α_r, the number of walks to simulate n_w, a relevance threshold t_s—and returns a subgraph T containing the relevant vertices in relation to q. This algorithm can be viewed in three major steps:

1. Compute the walk distribution around seed node q in graph B using random walk with restart (line 2). We omit the Random Walk algorithm due to space constraints, however, the technique is described above.

2. Determine the number of vertices to include in the subgraph T based on the relevance threshold parameter t_s, mean of the walk distribution list $\mu(L)$ and the standard deviation of the walk distribution list $\sigma(L)$ (lines 4-6).
3. Create a subgraph based on the included vertices (line 8).

Algorithm 1: Local Proximity

```
  Input: Graph $B$, seed node $q$, teleport value $\alpha_r$, number of walks to simulate
         $n_w$, relevance threshold $t_s$
  Result: Subgraph $T$
1 subgraph_nodes = [];
2 $D$ = RandomWalk($q$, $\alpha_r$, $n_w$, $B$);
3 $L$ = $D$.values;
4 for vertex $u$ in $B$ do
5     if $D[u] > \mu(L) + \sigma(L) / t_s$ then
6         subgraph_nodes.append(u);
7 $T$ = $B$.subgraph(subgraph_nodes);
8 return $T$;
```

AttriPart. Armed with the LocalProximity algorithm, we further propose an algorithm AttriPart, which takes into account the network structure and attribute information contained in graph to find denser local partitions than can be found using the network structure alone. The foundation of this algorithm is based on Spielman and Teng (2013)Andersen *et al.* (2006)Zhukov (2015) with subtle modifications on lines 1, 4 and 9. These modifications incorporate the addition of a combined graph model, approximate PageRank computation using the LocalProximity algorithm, and the parallel cut and conductance metric. In addition, AttriPart doesn't depend on reaching a target conductance in order to return a local partition—instead it returns the best local partition found within sweeping n_s vertices of the sorted PageRank vector.

Algorithm Description. Given a graph B, seed node $q \in V$, target conductance ϕ_o, rank truncation threshold ϵ, the number of iterations to compute the rank vector t_{last}, teleport value α_n, rank iteration threshold ϵ_t and number of nodes to sweep n_s—AttriPart will find a local partition S around q within n_s iterations of sweeping. This algorithm can be viewed in five steps:

1. Set values for ϵ and t_{last} as seen in Eq. (3.7) and Eq. (3.9) respectively. We experimentally set $b = \frac{1+log(m)}{2}$ and ϵ_t to 0.01. For additional detail on parameters ϵ, t_{last} and b see Spielman and Teng (2013). For all other parameter values see Section 3.4.
2. Run LocalProximity around seed node q in order to reduce the run time of the PageRank computations (line 1).
3. Compute the PageRank vector using a lazy random transition with personalized restart—with preference vector s containing all the probability on seed node q. At each iteration truncate a vertex's rank if it's degree normalized PageRank score is less than ϵ (lines 2-7).
4. Divide each vertex in the PageRank vector by its corresponding degree centrality and order the rank vector in descending order (line 8).
5. Sweep over the PageRank vector for the first n_s vertices, returning the best local partition S found (lines 9-10). The sweep works by taking the re-organized rank vector and creating a set of vertices S by iterating through each vertex in the rank vector one at a time, each time adding the next vertex in the rank vector to S and computing $\phi(S)$.

$$\epsilon = 1/(1800(l+2)t_{last}2^b) \tag{3.7}$$

$$l = \lceil log_2(2m/2) \rceil \tag{3.8}$$

$$t_{last} = (l+1)\lceil \frac{2}{\phi^2} ln(c_1(l+2)\sqrt{2m/2}) \rceil \tag{3.9}$$

Algorithm 2: AttriPart

```
Input: Graph $B$, seed node $q$, target conductance $\phi_o$, truncation threshold $\epsilon$,
       iterations $t_{last}$, teleport value $\alpha_n$, iteration threshold $\epsilon_t$, vertices to
       sweep $n_s$
Result: Local partition $S$
$T$ = Local_Proximity($B$, $q$, $\alpha_r$, $n_w$, $t_s$);
$D_{i,i} = \delta(v_i)$;
$W = \frac{1}{2}(I + D^{-1}T)$;
for $t = 1$ to $t_{last}$ and $sum(q_t)$ - $sum(q_{t-1})$ ¡ $\epsilon_t$ do
    $q_t = (1-\alpha)q_{t-1}W + \alpha s$;
    $r_t(i) = q_t(i)$ if $q_t(i)/d(i) > \epsilon$, else 0;
Order $i$ from large to small based on $r_t(i)/d(i)$;
Sweep Parallel_Conductance $\phi(S\{i = 1..j\})$ while $i < n_s$;
If there is j : $\phi(S_j) < \phi_o$, return $S$;
```

LocalForecasting. As a natural application of the AttriPart algorithm, we introduce a method to predict how local communities will evolve over time. This method is based on the AttriPart algorithm with two significant modifications—(i) required use of the LocalProximity algorithm to create a subgraph around the seed node and (ii) the use of the ExpandedNeighborhood algorithm to predict links between nodes in the subgraph. The idea behind using the ExpandedNeighborhood algorithm is that nodes are often missing many connections they will make in the future, which in turn affects the grouping of nodes into communities. To aid in predicting future edge connections we use Jaccard Similarity Liben-Nowell and Kleinberg (2007) to predict the likelihood of each vertex connecting to the others—with edges added if the similarity between two nodes is greater than threshold t_e.

Algorithm Description. Given a graph B, a seed node $q \in V$, a target conductance ϕ_o, a rank truncation threshold ϵ, the number of iterations to compute the rank vector t_{last}, a teleport value α_n, rank iteration threshold ϵ_t, similarity threshold t_e and number of nodes to sweep n_s—this algorithm will find a predicted local partition around q within n_s iterations of sweeping. As the LocalForecasting algorithm is similar to AttriPart, we highlight the three primary steps:

1. Determine the subgraph around a given seed node using the LocalProximity algorithm (line 1).
2. Determine the pairwise similarity between all nodes in the subgraph using Jaccard Similarity, adding edges that are above a given similarity threshold (line 2).
3. Run the AttriPart algorithm to find the predicted local partition around the seed node (line 3).

3.3.3 Analysis

Effectiveness. LocalProximity (Algorithm 1). The objective is to ensure that all relevant nodes in proximity to seed node q are included. We use the fact that many real-world graphs follow a scale-free distribution Barabási and Albert (1999) Faloutsos *et al.* (1999), with many nodes containing only a few links while a handful

```
Algorithm 3: Local Forecasting
  Input: Graph $B$, seed node $q$, target conductance $\phi_o$, truncation threshold $\epsilon$,
         iterations $t_{last}$, teleport value $\alpha_n$, iteration threshold $\epsilon_t$, similarity
         threshold $t_e$, vertices to sweep $n_s$
  Result: Predicted local partition $S$
  $T$ = Local_Proximity($B$, $q$);
  $T$ = Expanded_Neighborhood($T$, $t_e$) ;
  $S$ = AttriPart($T$, $q$, $\phi_o$, $\epsilon$, $t_{last}$, $\alpha_n$, $\epsilon_t$, $n_s$) ;
  return $S$ ;
```

```
Algorithm 4: Expanded Neighborhood
  Input: Subgraph $T$, edge addition threshold $t_e$
  Result: Subgraph $T$ with predicted edges
  for $u$ in $T$ do
      for $v$ in $T$ and $v$ not $u$ do
          u_attr = $T[u]$; v_attr = $T[v]$;
          similarity_score = JaccardSimilarity(u_attr, v_attr);
          if similarity_score $> t_e$ and not $T[u][v]$ then
              $T[u][v]$ = similarity_score;
  return $T$;
```

encompasses the majority. In Figure 3.4, we found that after running n_w trials of random walk with restart, a scale-free like distribution formed—with a large majority of the nodes containing a small number of 'hits', while a few nodes constituted the bulk.

As the number of random walks n_w is increased, the scale-free like distribution is maintained since each node is proportionally walked with the same distribution. We

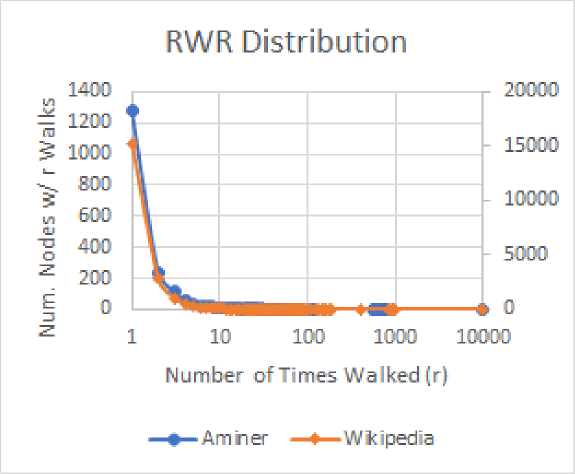


Figure 3.4: Random walk w/ restart—distribution of node walk counts. n_w = 10,000, α_r = 0.15; dataset: wikipedia, start vertex: 'ewok', y-axis: right; dataset: Aminer, start vertex: 364298, y-axis: left. We omit nodes walked zero times in the graph, however, they're used in calculating $\mu(L)$, $\sigma(L)$.

therefore need only some minimum value for n_w, which we set to 10,000. We use this skewed scale-free like distribution in combination with Eq. (3.10) below to ensure the extraction of relevant nodes in relation to a query vertex.

Mathematically we define node relevance based on Eq. (3.10), where D is a dictionary containing the walk count of each vertex and $D(v)$ represents the number of times vertex v is walked in n_w trials of the random walk with restart. L is a list of each node's walk count in the graph, $\mu(L)$ is the average number of times all of the nodes in the graph are walked and $\sigma(L)$ is the standard deviation of the number of times all of the nodes in the graph are walked. In section 3.4 we discuss values of t_s that have been shown to be empirically effective.

$$D(v) > \mu(L) + \sigma(L)/t_s \tag{3.10}$$

After determining the relevant nodes we create a subgraph T from a portion of the long-tail curve as defined by threshold parameter t_s in conjunction with $\mu(L)$ and $\sigma(L)$. We say that subgraph T contains $p \ll n$ nodes—with p increasing nearly independently of the graph size (depending on threshold t_s). As seen in Figure 3.4 the number of nodes with r walks converges independent of graph size.

Efficiency. All algorithms use the same data structure for storing the graph information. If a compressed sparse row (CSR) format is used, the space complexity is $O(2m+n+1)$. Alternatively, we note that with minor modification to the algorithms above we can use an adjacency list format with $O(n+m)$ space.

Lemma 3.3.0.1 (Time Complexity). LocalProximity *has a time complexity of* $O(n+m_p+n_w)$ *while* AttriPart *has a time complexity of* $O(p^2+pm_p+n+n_w)$ *and* LocalForecasting *a time complexity of* $O(p^2+pm_e+n+n_w)$.

Proof. LocalProximity: There are three major components to this algorithm: (1) n_w random walks with walk length l for a time complexity of $O(n_w)$ (line 2). (2) Linear iteration through the number of nodes taking $O(n)$ (lines 4-7). (3) Subgraph T creation based on the number of included vertices p with node set V_t—requiring iteration through every edge of node $v \in V_t$ for m_p total edges. Iterating through every edge is linear in the number of edges for a time complexity of $O(m_p)$ (line 8). This leads to a total time complexity of $O(n+m_p+n_w)$

AttriPart: There are six major steps to this algorithm: (1) calling LocalProximity which returns a subgraph T containing p nodes and m_p edges for a time complexity of $O(n+m_p+n_w)$ (line 1). (2) Creating a diagonal degree matrix by iterating through each node in T with time complexity $O(p)$ (line 2). (3) Creating the lazy random walk transition matrix W, which requires $O(m_p)$ from multiplying the corresponding matrix entries (line 3). (4) In lines 4-7 we iterate for t_{last} iterations, with each iteration (i) updating the rank vector by multiplying the corresponding edges in the transition matrix W, with the rank vector q for a time complexity of $O(m_p)$ and (ii) truncating every vertex with rank $q_t(i)/d(i) \leq \epsilon$ for a time complexity linear in the number of nodes in the rank vector $O(p)$. (5) Sort the rank vector which will be upper bounded by $O(plogp)$ (line 8). (6) Compute the parallel conductance, which takes $O(p^2 + pm_p)$ time (lines 9-10). Combining each step leads to a total time complexity of $O(p^2 + pm_p + n + n_w)$.

LocalForecasting: This algorithm has three major steps: (1) run the LocalProximity algorithm, which has a time complexity of $O(n + m_p + n_w)$. (2) Perform the ExpandedNeighborhood algorithm, which densifies T by adding predicted edges for a total of m_e edges in T. This algorithm has a time complexity of $O(p^2)$ due to the nested for loops. (3) Run the AttriPart algorithm, which has a time complexity of $O(p^2 + pm_e + n + n_w)$ with the modification of m_p to m_e for the additional edges. This leads to an overall time complexity of $O(p^2 + pm_e + n + n_w)$. □

While AttriPart and LocalForecasting both scale quadratically with respect to p, we note that in practice these algorithms are very fast since $p \ll n$ and p scales nearly independent of graph size as shown in section 3.3.3.

3.4 Experiments

In this section, we demonstrate the effectiveness and efficiency of the proposed algorithms on three real-world network datasets of varying scale.

3.4.1 Experiment setup

Datasets. We evaluate the performance of the proposed algorithms on three datasets—(1) the Aminer co-authorship network Zhang *et al.* (2017), (2) a Musician network mined from DBpedia and (3) a subset of Wikipedia entries in DBpedia containing both abstracts and links. All three networks are undirected with detailed information on each below:

- Aminer. Nodes represents an author, with each author containing a set of topic keywords, and an edge representing a co-authorship. To form the attribute network, we compute attribute edges based on the similarity between two authors for every network edge, using Jaccard Similarity on the corresponding authors's topic set.

- Musician. Nodes represent a Musician, with each Musician containing a set of music genres, and an edge representing two Musicians who have played in the same band. To form the attribute network, we compute attribute edges based on the similarity between two Musicians for every network edge, using Jaccard Similarity on the corresponding artist's music genre set.

- Wikipedia. Nodes represent an entity, place or concept from Wikipedia which we will jointly refer to as an item. Each item contains a set of defining key words; with edges representing a link between the two items. The dataset originates from DBpedia as a directed graph with links between Wikipedia entries. We modify the graph to be undirected for use with our algorithms—which we believe to be a reasonable as each edge denotes a relationship between two items. In addition, this dataset uses only a portion of the Wikipedia entries containing both abstracts and links to other Wikipedia pages found in DBpedia. To form the attribute network, we compute attribute edges based on the similarity between two items for every network edge using Jaccard Similarity on the corresponding item's key word set.

Metrics. (1) To benchmark the LocalProximity algorithm's effectiveness and efficiency, we compare (i) the difference between local partition created with and

Category	Network	Nodes	Edges
Aminer	Co-Author	1,560,640	4,258,946
Musician	Co-Musician	6,006	8,690
Wikipedia	Link	237,588	1,130,846

Table 3.2: Network Statistics

without the LocalProximity algorithm on AttriPartand (ii) the run time and difference between the top 20 PageRank vector entries with and without the LocalProximity algorithm. (2) To benchmark the AttriPart algorithm's effectiveness and efficiency we compare the triangle count, node count, local partition density and run time to PageRank-Nibble. Normally, PageRank-Nibble does not return a local partition if the target conductance is not met, however, we modify it to return the best local partition found—even if the target conductance is not met. This modification allows for more comparable results to AttriPart. (3) To provide a baseline for the LocalForecasting algorithm's effectiveness, we compare the local partition results to AttriPart on two graph missing 15% of their edges.

Use-Case. While not covered in this thesis, we note that an interesting extension of this research would be to create a use-case study comparing various local partitioning algorithms on networks with ground truth data. This would allow for a more in-depth study of the true effectiveness of the proposed local partitioning algorithm compared to baseline local partitioning techniques.

Repeatability. All data and source code used in this research will be made publicly available. The Aminer co-authorship network can be found on the Aminer website [2] ; the Musician and Wikipedia datasets used in the experiments will be released on

[2] https://Aminer.org/data

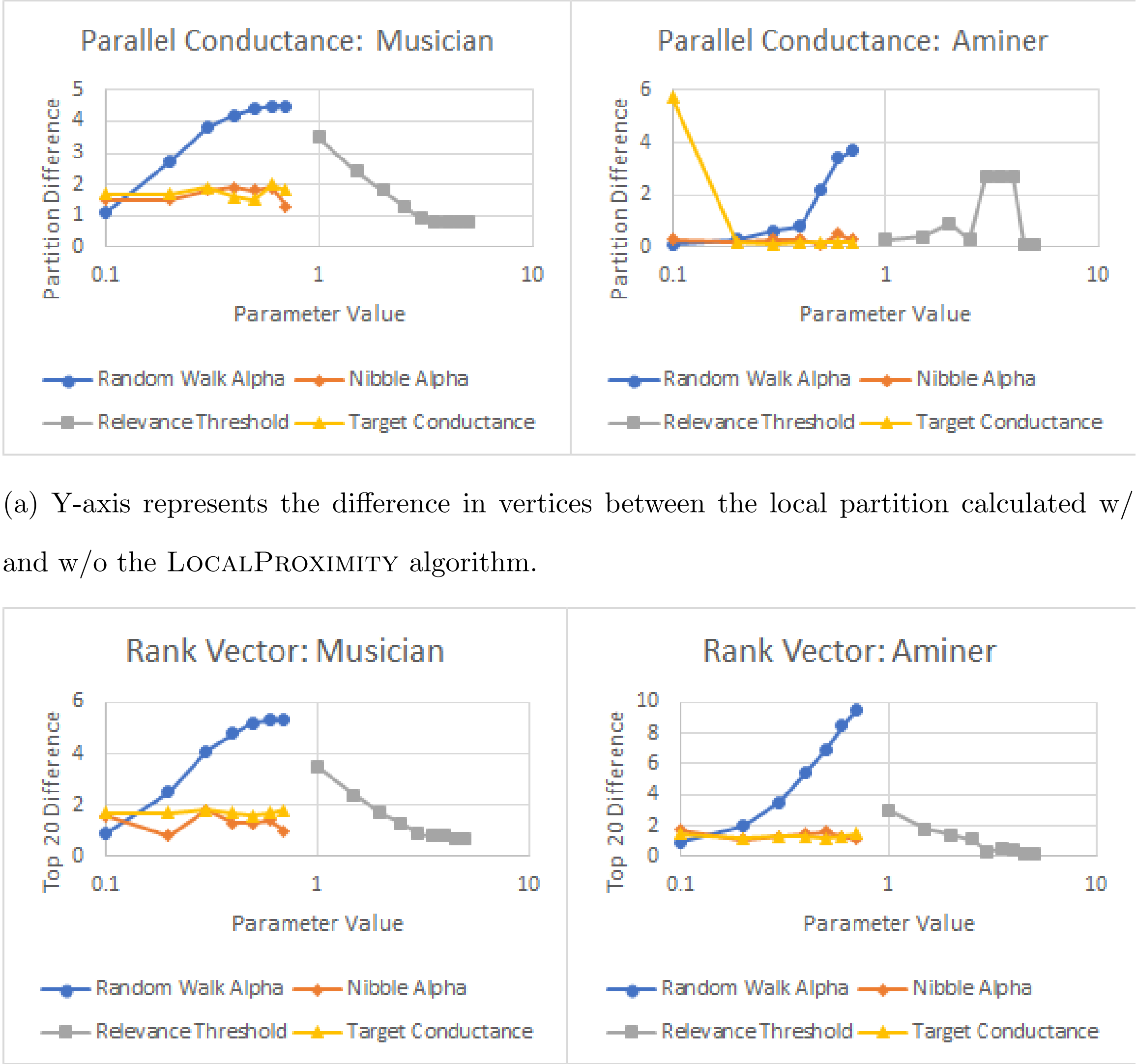
the author's website. All algorithms and experiments were conducted in a Windows environment using Python.

3.4.2 Effectiveness

LocalProximity. In Figure 3.5 parts (a)-(c), we can see that the proposed LocalProximity algorithm significantly reduces the computational run time, while maintaining high levels of accuracy across both metrics. Parts (a)-(b) demonstrate to what extent the accuracy of the results are dependent upon the parameter values. In particular, a low value of α_r (random walk alpha) and a high value of t_s (relevance threshold) are critical to providing high accuracy results.

In Figure 3.5 part (a), we measure accuracy as the number of vertices that differ between the local partitions w/ and w/o the LocalProximity algorithm on AttriPart. A small partition difference indicates that the LocalProximity algorithm finds a relevant subgraph around the given seed node and that the full graph is unnecessary for accurate results. In part (b), we define the accuracy of the results to be the difference between the set of top *20* entries in the PageRank vectors for the full graph and subgraph using the LocalProximity algorithm. Overall, the results from part (b) correlate well to (a)—showing that for low values of α_r (random walk alpha) and high values of t_s (relevance threshold), their is negligible difference between the results computed on the full graph and the subgraph found using the LocalProximity algorithm.

AttriPart. In Figure 3.7, we see that AttriPart finds significantly denser local partitions than PageRank-Nibble—with local partition densities approximately 1.6×, 1.3× and 1.1× higher in AttriPart than PageRank-Nibble in the Aminer, Wikipedia and Musician datasets respectively. Density is measured as $\frac{2m}{n(n-1)}$ where m is the number of edges and n is the number of nodes.



(a) Y-axis represents the difference in vertices between the local partition calculated w/ and w/o the LocalProximity algorithm.

(b) Y-axis represents the # of vertices differing between the top 20 rank vector entries w/ and w/o the LocalProximity algorithm.

Figure 3.5: Each data point averages 10 randomly sampled vertices in both the Aminer and Musician datasets. Default parameters (unless sweeped across): $\alpha_n = 0.2$, $\alpha_r = 0.15$, $\phi_o = 0.2$, $t_s = 2$, $n_w = 10{,}000$, $n_s = 200$. Parameter ranges: α_r, α_n and ϕ_o [0.1-0.7] in 0.1 intervals; t_s [1-5] in 0.5 intervals.

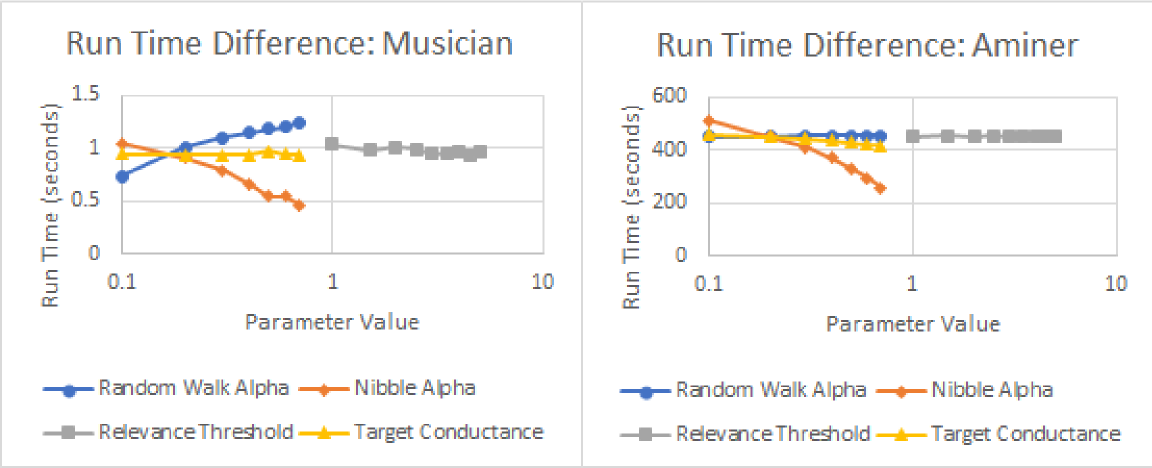
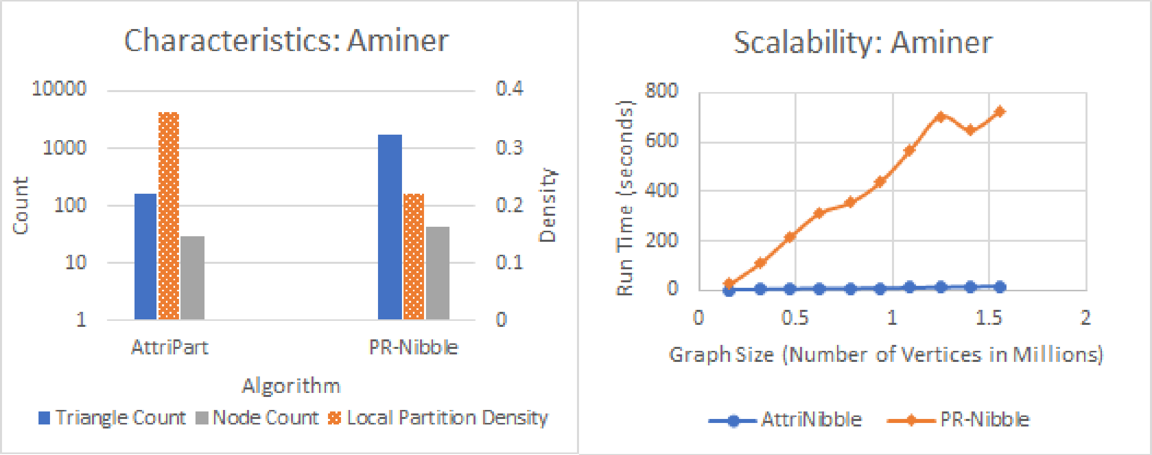


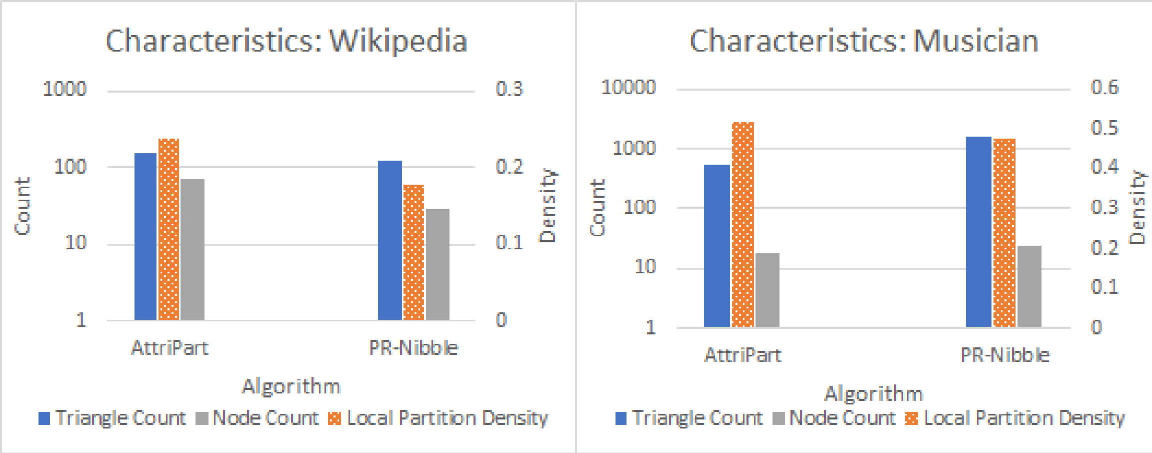
Figure 3.6: Y-axis represents the difference in run time between the PageRank calculation w/ and w/o the LocalProximity algorithm. Each data point averages 10 randomly sampled vertices in both the Aminer and Musician datasets. Default parameters (unless sweeped across): $\alpha_n = 0.2$, $\alpha_r = 0.15$, $\phi_o = 0.2$, $t_s = 2$, $n_w = 10{,}000$, $n_s = 200$. Parameter ranges: α_r, α_n and ϕ_o [0.1-0.7] in 0.1 intervals; t_s [1-5] in 0.5 intervals.

In Figure 3.7, we observe that the triangle count of the AttriPart algorithm is lower than PageRank-Nibble in the Musician and Aminer datasets. We attribute this to the fact that AttriPart is finding smaller partitions (as measured by node count) and, therefore, there are less possible triangles. While no sweeps across algorithm parameters were performed, we believe that the gathered results provide an effective baseline for parameter selection. We also note that ideally the we would have ground truth data for the local partition in each graph, however, in their absence we use the graph characteristics described in Figure 3.7.

LocalForecasting. In order to measure the effectiveness of the LocalForecasting algorithm we setup the following experiment with three local partition calculations: (1) calculate the local partition using AttriPart, (2) calculate the local partition using AttriPart with 15% of the edges randomly removed from the graph and (3) calculate the local partition using the LocalForecasting algorithm with



(a) Scalability: Each data point represents the Aminer dataset in 1/10th intervals, with each point averaged over 3 randomly sampled vertices. Parameters: $\alpha_n = 0.2$, $\alpha_r = 0.15$, $\phi_o = 0.2$, $t_s = 2$, $n_w = 10{,}000$, $n_s = 200$.



(b)

Figure 3.7: Characteristics: results are averaged over 20 and 100 randomly sampled vertices in the Aminer/Wikipedia and Musician datasets, respectively. Parameters: $\alpha_n = 0.2$, $\alpha_r = 0.15$, $\phi_o = 0.05$, $t_s = 2$, $n_w = 10{,}000$, $n_s = 200$.

15% of the edges randomly removed from the graph. We treat (1) as the baseline local community and want to test if (3) finds better local partitions than (2). The idea behind randomly removing 15% of the edges in the graph is to simulate the evolution of the graph over time and test if the LocalForecasting algorithm can predict better local communities in the future. Ideally, we would have ground-truth local community data for a rich graph with time series snapshots, however, in its absence we use the above method.

In Figure 3.8, each data point is generated in three steps—(i) taking the difference between the set of vertices and edges in local partitions (1) and (3), (ii) taking the difference between the set of vertices and edges in local partitions (1) and (2) and (iii) by taking the difference between (ii) and (i). Step (i) tells us how far off the LocalForecasting algorithm is from the baseline, step (ii) tells us how far off the local partition would be from the baseline if no prediction techniques were used and step (iii) tells us the difference between the local partitions with and without the LocalForecasting algorithm (which is what we see graphed in Figure 3.8).

In Figure 3.8, we see that the local partition prediction accuracy, for both the edges and vertices, is above the baseline calculations in the Aminer dataset for a majority of edge similarity threshold values (t_e). The best results were obtained when t_e is 0.6, with an average of 1.4 vertices and 2.75 edges predicted over the baseline using the LocalForecasting algorithm. This number, while relatively small, is an average of 20 randomly sampled vertices—with one result reaching up to 14 vertices and 26 edges over baseline. In addition, we can see that the Musician dataset does not perform as well as the Aminer dataset, with most of the prediction results performing worse than the baseline (as indicated by the negative difference). We believe that this result on the Musician dataset is due to the different nature of each dataset's network structure—with the Musician dataset being significantly more sparse (no giant connected component) than the Aminer dataset.

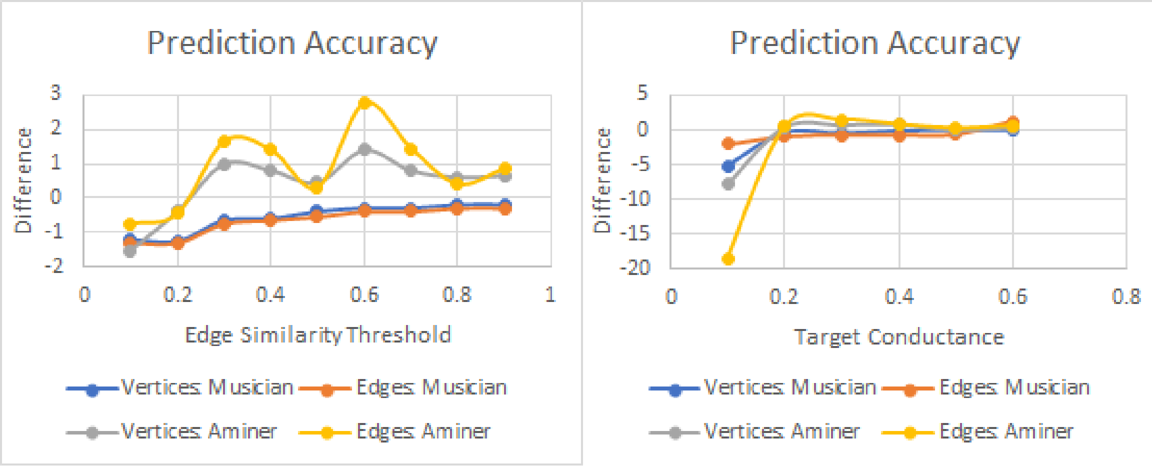


Figure 3.8: Each data point averages 20 randomly sampled vertices in the Aminer and Musician datasets. Default parameters (unless sweeped across): $\alpha_n = 0.2$, $\alpha_r = 0.15$, $\phi_o = 0.2$, $t_s = 5$, $t_e = 0.7$, $n_w = 10{,}000$, $n_s = 200$. Parameter ranges: t_e [0.1-0.9] in 0.1 intervals, ϕ_o [0.1-0.6] in 0.1 intervals.

3.4.3 *Efficiency*

For both the proposed and baseline algorithms, the efficiency results represent only the time taken to run the algorithm (e.g. not including loading data into memory). LocalProximity. Across a majority of the parameters the run time for the full graph PageRank computation is approximately 450 seconds longer compared to computing the PageRank vector based on the LocalProximity sugraph. AttriPart. In Figure 3.7, we see that the AttriPart algorithm finds local partitions 43× faster than PageRank-Nibble. LocalForecasting. This algorithm has an expected run time nearly identical to AttriPart, we therefore refer the reader to Figure 3.7 for run time results.

3.5 Use-Cases

We propose five use-cases as a future extension of this research in the multi-network domain, where we are given two graphs containing both data and knowledge information. We say that the knowledge layer contains relationships between concepts and that the data layer contains relationships between people or items. The use cases are as follows: (1) given a person, we want to provide explainable predictions for the evolution of the region around this person utilizing both the knowledge and data information; (2) given a person, we want to find the relevant topics or concepts in relation to them; (3) given a topic, we want to find the relevant people in relation to the topic; (4) given a topic and a person, we want to measure the similarity between the topic and person; and (5) given two or more people/topics, we want to create a similarity ranking between the topics and users.

3.6 Conclusion

This paper proposes new algorithms for attributed graphs, with the goal of (i) computing denser local graph partitions and (ii) predicting the evolution of local communities. We believe that the proposed algorithms will be of particular interest to data mining researchers given the computational speed-up and enhanced dense local partition identification. The proposed local partitioning algorithm AttriPart has already deployed to the web platform PathFinder (www.path-finder.io) Freitas *et al.* (2017) and allow users to interactively explore all three datasets presented in the paper. In addition, the source code and datasets will be made publicly available.

Chapter 4

RAPID ANALYSIS OF NETWORK CONNECTIVITY

This research focuses on accelerating the computational time of two base network algorithms (k-simple shortest paths and minimum spanning tree for a subset of nodes)—cornerstones behind a variety of network connectivity mining tasks—with the goal of rapidly finding network pathways and trees using a set of user-specific query nodes. To facilitate this process we utilize: (1) multi-threaded algorithm variations, (2) network re-use for subsequent queries and (3) a novel algorithm, Key Neighboring Vertices (KNV), to reduce the network search space. The proposed KNV algorithm serves a dual purpose: (a) to reduce the computation time for algorithmic analysis and (b) to identify key vertices in the network (context). Empirical results indicate this combination of techniques significantly improves the baseline performance of both algorithms. We have also developed a web platform utilizing the proposed network algorithms to enable researchers and practitioners to both visualize and interact with their datasets (PathFinder: `http://www.path-finder.io`).

4.1 Introduction

Motivation. With the advent of the big data era and the emergence of network science, large-scale networks are appearing across many disciplines, from medicine and epidemiology to advertising and marketing. As a result, an exponential amount of network data is being generated at an unprecedented rate. The challenge before us, given limited computational resources, is to translate this large network data into meaningful knowledge.

Problem. How can we rapidly explore, analyze and visualize a set of user-specific query nodes in relation to a dataset? We envision that tree, pathway and context are the three key components to answer this question. Formally, given a graph $G = (V, E)$ and a set of user-specific query nodes Q, we seek to find (1) the relationship between each query node in Q (i.e. tree detection), (2) a subset of paths $R \subset G$ such that R contains only vertices and edges that provide key path information between different query nodes in Q (i.e. pathway detection) and (3) a subset of important vertices, C and edges, S, such that $C \subset V$ and $S \subset E$ (i.e. context detection).

Contributions. Our main contributions are three-fold: (1) the development of two multi-threaded algorithms: k-simple shortest paths (KSSP) and minimum spanning tree for a subset of nodes (Shortest Paths MST); (2) the creation of a novel algorithm, Key Neighboring Vertices (KNV), for reducing network search space and identifying key vertices; (3) the development of a web platform, utilizing these algorithms, for researchers and practitioners to upload their own network data for analysis and visualization.

1. K-Simple Shortest Paths (pathway): Our approach to the k-simple shortest path algorithm offers three new features: (a) instead of creating a multi-threaded single source shortest path algorithm Ruppert (2012)Guerriero and Musmanno (2000)Singh and Singh (2015), we parallelize each single source shortest path computation required to find a path in k (for details see section 3.1); (b) we pre-process the network data to reduce the search space using the KNV algorithm; (c) the generated pathways and surrounding context nodes can be re-queried to find additional paths or trees.

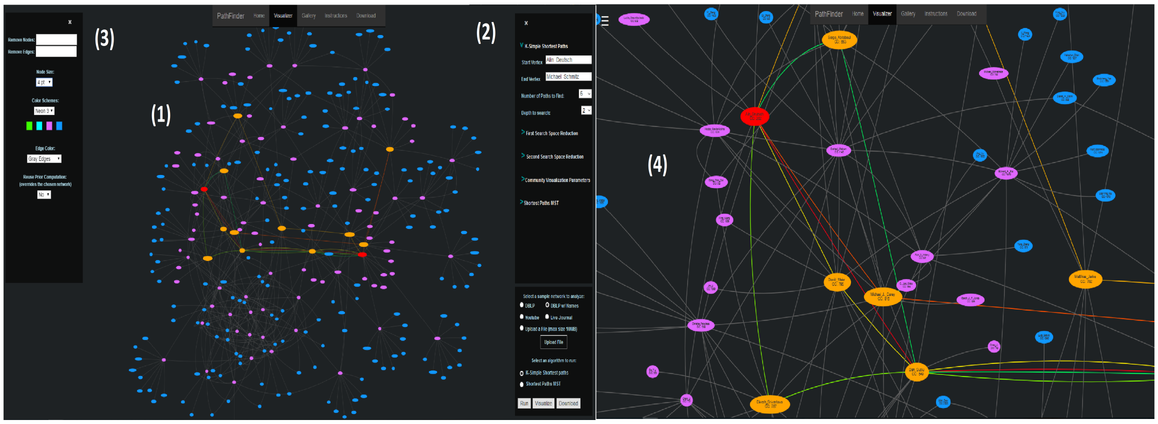
Shortest Paths MST (tree): Our approach to solving the minimum spanning tree for a subset of nodes (Shortest Paths MST) is centered around (a) parallelizing the shortest path computations to run simultaneously; (b) pre-processing the network data to reduce the search space of the algorithm; (c) re-querying the generated tree and surrounding context nodes to find additional trees or paths.

2. Key Neighboring Vertices (context): Our approach is similar to Sulieman et. al., but with three key differences. (a) Instead of exploring the network based on the top N vertices with highest degree centrality, we seed the map with a set of user-specific query nodes; (b) we propose using above average network degree centrality as the metric for including nearby neighboring vertices for further exploration in the search list; (c) we allow additional parameters that let the user control the exploration process.

3. Platform: We have developed PathFinder, a web platform to assist users in mining network connectivity from large networks. PathFinder begins by taking an input network uploaded by the user or selection from a pre-loaded dataset. Depending on the user's expertise with the program there are two sets of controls: basic and advanced. The basic controls allow the user to start without an understanding of the algorithms, while the advanced controls allow the user to fine-tune their queries and obtain information that may not be available with a basic search. Visualizations are generated using vis.js and GraphViz. A video demo of the platform is available at: `https://youtu.be/PxQVd-6mKUw`.

4.2 Platform Functionality

Each part of Figure 4.1 highlights some of the platforms core functionality. Figure 4.1(1) shows a sample visualization using the pathway detection algorithm on the DBLP network. Figure 4.1(2) allows the user to enter the algorithm parameters and select a network for analysis. Figure 4.1(3) allows the user to enter nodes and edges to be removed from the graph search, select whether or not to re-use the current graph results for further analysis and change the configuration of network style parameters, including: node-edge color scheme and node size. Figure 4.1(4) is a zoomed in portion of Figure 4.1(1). The red nodes represent the start and end query vertices, orange



Remove Nodes:

Remove Edges:

Node Size: 4 pt

Color Schemes: Neon 3

🟩 🟦 🟪 🟦

Edge Color: Gray Edges

Reuse Prior Computation: (overrides the chosen network) No

K-Simple Shortest Paths

Start Vertex: Alin Deutsch

End Vertex: Michael Schmitz

Number of Paths to Find: 6

Depth to search: 2

First Search Space Reduction

Second Search Space Reduction

Community Visualization Parameters

Shortest Paths MST

Select a sample network to analyze:

● DBLP ○ DBLP w/ Names

● Youtube ● Live-Journal

● Upload a File (max size 10MB)

Select an algorithm to run:

○ K-Simple Shortest paths

● Shortest Paths MST

Figure 4.1: An illustrative example of our platform to find the key pathways. Start and end vertices are in red.

for intermediate path vertices, blue for one hop away critical vertices and purple for two hop away critical vertices. The blue and purple vertices surrounding the the path vertices are determined by the KNV algorithm.

Engaging the Audience. We expect that our demo will primarily attract two audiences, (1) practitioners who are interested in exploring the connectivity between key nodes in large networks, and (2) information management and data mining researchers who develop new algorithms and tools.

4.3 Technical Details

All algorithms perform on an undirected, unweighted, adjacency list graph representation, $G = (V, E)$. Nonetheless, we note that the proposed platform is flexible to admit alternative algorithms and graph types with mild changes.

4.3.1 Pathway Detection

Problem definition. Given two pre-marked vertices, $x, y \epsilon V$ from the graph $G = (V, E)$, this algorithm will find k-simple shortest paths from x to y.

Algorithm description. We adopt the k-simple shortest paths threaded and search reduced algorithm (Algorithm 5) to detect key pathways that connect the query nodes, which can be viewed in 7 steps:

1. Run the Key Neighboring Vertices (KNV) algorithm to reduce the search space of the graph using the start and end vertices.
2. Find the first single source shortest path between vertex x and vertex y.
3. Run the KNV algorithm a second time on the original graph with all the vertices from the shortest path.
4. For each edge in the current shortest path: temporarily remove the given edge and run a single source shortest path algorithm. Each of the shortest paths run in parallel.
5. Determine which of these paths produced the subsequent shortest path. Permanently delete the edge that formed the shortest path from the adjacency list.
6. Repeat Steps 4-5 until k shortest paths have been found or there are no more identifiable paths.
7. Optional: Re-query the generated network for additional paths, paths between different vertices or for a tree using Shortest Paths MST.

4.3.2 Tree Detection

Problem definition. Given two or more pre-marked vertices, $2...x \epsilon V$ from the graph $G = (V, E)$, this algorithm will find a MST that is constructed from a combination of single source shortest paths.

Algorithm description. We adopt the Shortest Paths Minimum Spanning Tree algorithm (Algorithm 6) to determine the relationship between the user-specified query nodes. This can be viewed in 4 steps:

Algorithm 5: Pathway Detection: K-Simple Shortest Paths Threaded and Search Reduced

```
Input: Graph A = (V, E); sv, ev ϵ V
Output: Array of paths: sPaths[]
Initialization: sPaths[], pHolder[], eHolder[]; cPath := 0
Graph B = Key_Neighboring_Vertices(A, sv, ev)
sPaths[cPath] = Dijkstra(sv, ev, B)
cPath++
Graph B = Key_Neighboring_Vertices(A, sPaths)
while cPath < numPaths do
    pIndex := 0
    for each edge e ϵ sPaths[cPath-1] do
        Graph C = B
        C.removeEdge(e)
        eHolder[pIndex] = e
        pHolder[pIndex] = new_thread(Dijkstra(sv, ev, C))
        pIndex++
    shortestPaths[cPath] = pHolder.getMinPath()
    B.removeEdge(eHolder[cPath])
    cPath++
```

1. Each pre-marked node, v, will run a single source shortest path algorithm against every other pre-marked vertex. The single source shortest path algorithms run in parallel.
2. Sort the resulting paths in ascending order of path length.
3. Run Kruskal's algorithm to determine which of these shortest paths form the

MST.

4. Optional: Re-query the generated network with different vertices or find paths using the pathway detection algorithm.

```
Algorithm 6: Tree Detection: Shortest Paths MST
Input: Graph A = (V, E); Array of integers: vertices[]
Output: Array of paths: sPaths[]
Initialization: struct PathInfo { Vertex v1, v2 }, PathInfo paths[], Two
  integers: pathCount, pathsFound := 0
for each unique pair of vertices p1, p2 $\epsilon$ vertices do
    paths[pathCount].v1 = p1, paths[pathCount].v2 = p2
    pathCount++
Graph B = Key_Neighboring_Vertices(A, vertices)
for i := 0 to pathCount do
    sPaths[pathsFound] = new_thread(Dijkstra(B, paths[i]))
    pathsFound++
Sort_Ascending_Order(sPaths)
Kruskal's Algorithm(sPaths)
```

4.3.3 Context & Speed-up

For both tree and pathway detection, we propose an efficient algorithm to detect key neighboring vertices to reduce the search space using a combination of three techniques to identify critical nodes: (1) vertex centrality, (2) edge connection to a pre-marked node and (3) breadth first search. This allows us to create a reduced graph $R = (V, E)$, that is a subset of $G, R \subset G$. Through this process we implicitly assume that vertices with high centrality are key hubs in the graph and are therefore important ‘players’ in the network.

The proposed key neighboring vertices algorithm can be viewed in 4 steps:

1. Determine if the current vertex has an edge connection to one of the ‘key’ vertices and has above average vertex centrality. If both conditions are met, place the current vertex into a bin of that key vertex.
2. Sort each bin in descending order of vertex centrality.
3. From each bin, take the top ‘x’ neighboring nodes as important vertices at that depth level and add them to the reduced adjacency list.
4. From each bin, a percentage of the top ‘x’ nodes will become ‘key’ vertices and recursively undergo the process until the specified depth level is reached.

Algorithm 7: Context & Speed-up: Key Neighboring Vertices

```
Input: Graph A = (V, E); Array of ints: vertices[]; Six ints: cDepth, depth,
        avgCentrality, numVertex, numVerticesNextIter, numVerticesCritical
Output: Graph R = (V, E)
Initialization: bins[][]
for i := 0 to A.size do
    if A[i].centrality > avgCentrality then
        for k := 0 to vertices.size do
            if vertices[k] ∩ A[i] then
                bins[k] += i
for each array, a, in bins do
    Sort_Descending_Order_Vertex_Centrality(a);
for i := 0 to vertices.size do
    for k := 0 to bins[i].size and k ¡ numVertex do
        if !vertices[i] ∩ R[bins[i].at(k)] then
            R.addEdge(vertices[i], bins[i].at(k))
if cDepth < depth then
    numVertex = numVerticesNextIter
    vertices = new vertices[]
    cDepth++
    for each array, a, in bins do
        for i := 0 to a.size and i < numVerticesCritical do
            vertices.insert(a[i]])
    key_Neighboring_Vertices(vertices, numVertex, cDepth)
```

4.3.4 Empirical Evaluation

We used the DBLP co-authorship and the LiveJournal social network from the Stanford SNAP network to gather empirical data on the platform. The two measures we aim to quantify are speed and accuracy. Since the Shortest Paths MST and K-Simple Shortest Paths algorithm utilize the same search space reduction algorithm and multithreading techniques, we use the KSSP algorithm to represent the Shortest Paths MST in terms of accuracy and run time. All data was collected locally and does not account for any additional run time caused by using the web-platform.

To compare the accuracy and run time, we ran three variations of the K-Simple Shortest Path algorithm (KSSP). The first variation (v.1) contained only the core KSSP algorithm with no search space reduction or multithreading. The second variation (v.2) ran the KSSP algorithm with multithreading (KSSPT). The third variation (v.3) ran the KSSP algorithm with multithreading and the search space reduction algorithm (KSSPR). The run time and accuracy of the three KSSP variations can be seen in Figure 4.2 and Figure 4.3-4.4 respectively. It should be noted that (v.1) and (v.2) of the KSSP algorithm will always find the shortest paths available, while the same guarantee cannot be extended to (v.3). The reasoning behind the possible sub-optimal path(s) for (v.3) is due to the nature of the search space reduction algorithm applied to the graph. The KNV algorithm uses a tradeoff between accuracy and run time, which can be varied depending on the parameters. In trials for Figure 4.2 we applied parameters to the KNV algorithm that retained accuracy at the cost of speed. However, even with this additional 'cost', trial one results show (v.3) over 2.5x faster than (v.1) and 1.7x faster than (v.2) with no loss of accuracy with respect to the full network (last data point). It can be seen in Figure 4.4 that the run time and path length for the variations is dependent upon the start and end vertices.

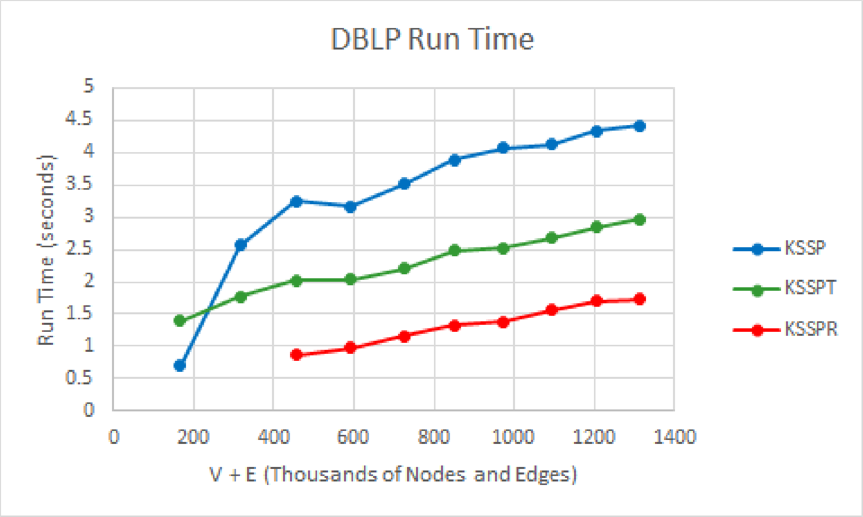


Figure 4.2: Start Vertex: 61, End Vertex: 70591, # of paths: 6. No data for KSSPR on first two data points due to selected KNV parameters.

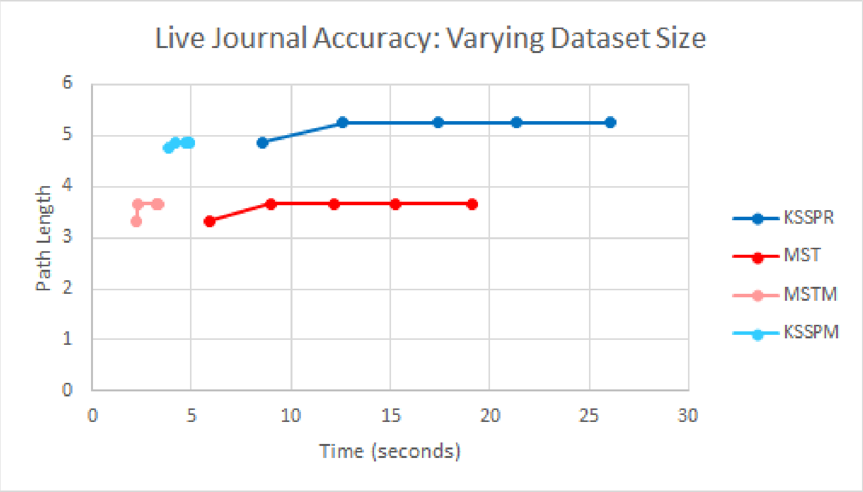


Figure 4.3: Data points represent 20%-100% of the Live-Journal network in 1/5th intervals. KSSPR: Start Vertex: 35521, End Vertex: 286345, # of paths: 8. MST Shortest Paths: Vertices: 0, 58, 9558, 34343.

In order to better access the abilities of the platform we ran both the KSSPR and Shortest Paths MST algorithms on the LiveJournal network. To put it in perspective, the LiveJournal network has approximately 38.5 million edges and vertices

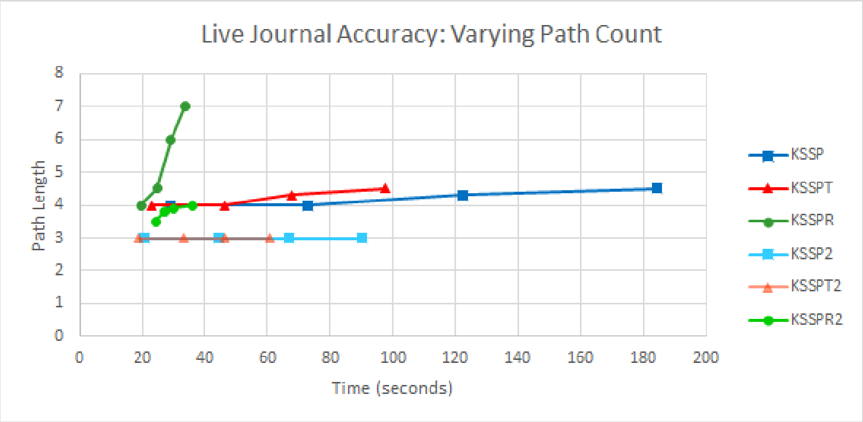


Figure 4.4: Each data point represents # of paths found: 2-14 in intervals of 4. Trial 1: Start Vertex: 35521, End Vertex: 286345. Trial 2: Start Vertex: 9790, End Vertex: 26073.

compared to the DBLP network of 1.3 million. In figure 4.3, the labels KSSPM and MSTM represent re-query results compared to Algorithm 5 and 6 with no re-query. Comparing the full network (5th interval) run time for KSSPR vs KSSPM we see an 5.3x speed-up and a 5.6x speed-up for the MSTM vs MST. It should be noted that (a) parameters were held constant when re-querying the network and that only information generated from the previous run is analyzed when re-querying and (b) shorter paths can be found when re-querying the graph since it's saved based on visualization parameters not search space reduction.

4.4 Conclusions

The goal of this work is to rapidly analyze network connectivity. We believe the computational speedup obtained will be of interest to information management and data mining researchers. In addition, the web platform PathFinder allows users to quickly and intuitively determine network connectivity between a set of user-specific query nodes. An operational prototype is online: `http://path-finder.io` and source code will be made publicly available.

Chapter 5

CONCLUSION AND FUTURE WORK

This thesis proposes three algorithms for analyzing the network connectivity of large scale graphs—with the goal of being able to analyze the connectivity between any number of seed nodes. In addition to addressing fundamental research questions regarding the connectivity analysis of marked nodes in graphs, we apply this research to address the problem of local community evolution and recommender systems in graphs. We believe that the proposed algorithms will be of particular interest to data mining researchers and practitioners given both the computational speed-up compared to traditional methods and their practical applications. All three algorithms are deployed to the web platform PathFinder (www.path-finder.io) Freitas *et al.* (2017) and allow users to interactively explore all the datasets presented. In addition, the source code and datasets will be made publicly available on the author's website.

We believe that there are many interesting extensions of this research in the areas of (1) multi-networks, (2) attributed algorithms for analyzing graphs containing two or more marked nodes, (3) exploration of additional link prediction techniques, (4) using the network structure of the graph itself to generate additonal attributes and (5) support for heterogeneous edges in the network analysis. Each of the aforementioned research areas has the potential to provide significant advances in the current methodology used to analyze network connectivity. In particular, we believe that extending this work to multi-networks would provide many interesting use cases for local community evolution and recommender systems.

REFERENCES

Akoglu, L., D. H. Chau, J. Vreeken, N. Tatti, H. Tong and C. Faloutsos, *Mining Connection Pathways for Marked Nodes in Large Graphs*, pp. 37–45 (2013), URL http://epubs.siam.org/doi/abs/10.1137/1.9781611972832.5.

Andersen, R., F. Chung and K. Lang, "Local graph partitioning using pagerank vectors", in "2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)", pp. 475–486 (2006).

Bagrow, J. P. and E. M. Bollt, "Local method for detecting communities", Phys. Rev. E URL https://link.aps.org/doi/10.1103/PhysRevE.72.046108 (2005).

Barabási, A.-L. and R. Albert, "Emergence of scaling in random networks", Science URL http://science.sciencemag.org/content/286/5439/509 (1999).

Cenek, P. and M. Hrcka, "Minimum spanning tree on a subset of nodes 63 minimum spanning tree on a subset of nodes", (2015).

Chen, J., O. Zaane and R. Goebel, "Local community identification in social networks", in "2009 International Conference on Advances in Social Network Analysis and Mining", pp. 237–242 (2009).

D. Sulieman, H. K. D. L., M. Malek, "Semantic social breadth-first search and depth-first search recommendation algorithms", URL http://lipn.fr/marami12/actes/sulieman.pdf (2013).

Dupont, P., J. Callut, G. Dooms, J. N. Monette and Y. Deville, "Relevant subgraph extraction from random walks in a graph", (2017).

Faloutsos, C., K. S. McCurley and A. Tomkins, "Fast discovery of connection subgraphs", in "Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining", KDD '04, pp. 118–127 (ACM, New York, NY, USA, 2004), URL http://doi.acm.org/10.1145/1014052.1014068.

Faloutsos, M., P. Faloutsos and C. Faloutsos, "On power-law relationships of the internet topology", in "Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication", SIGCOMM '99, pp. 251–262 (ACM, New York, NY, USA, 1999), URL http://doi.acm.org/10.1145/316188.316229.

Freitas, S., H. Tong, N. Cao and Y. Xia, "Rapid analysis of network connectivity", in "Proceedings of the 2017 ACM on Conference on Information and Knowledge Management", CIKM '17, pp. 2463–2466 (ACM, New York, NY, USA, 2017), URL http://doi.acm.org/10.1145/3132847.3133170.

Grolmusz, V., "A note on the pagerank of undirected graphs", Inf. Process. Lett. pp. 633–634, URL http://dx.doi.org/10.1016/j.ipl.2015.02.015 (2015).

Guerriero, F. and R. Musmanno, “Parallel asynchronous algorithms for the k shortest paths problem”, Journal of Optimization Theory and Applications pp. 91–108, URL https://doi.org/10.1023/A:1004676705907 (2000).

Haveliwala, T. H., “Topic-sensitive pagerank: a context-sensitive ranking algorithm for web search”, IEEE Transactions on Knowledge and Data Engineering (2003).

Hsu, C.-C., Y.-A. Lai, W.-H. Chen, M.-H. Feng and S.-D. Lin, “Unsupervised ranking using graph structures and node attributes”, in “Proceedings of the Tenth ACM International Conference on Web Search and Data Mining”, WSDM ’17, pp. 771–779 (ACM, New York, NY, USA, 2017), URL http://doi.acm.org/10.1145/3018661.3018668.

Jaewon yang, J. L., Julian McAuley, “Community detection in networks with node attributes”, ICDM (2013).

Kannan, R., S. Vempala and A. Vetta, “On clusterings: Good, bad and spectral”, J. ACM pp. 497–515, URL http://doi.acm.org/10.1145/990308.990313 (2004).

Laura Bennett, S. L. L. G. P. S. T., Aristotelis Kittas, “Community structure detection for overlapping modules through mathematical programming in protein interaction networks”, PLOS ONE URL https://doi.org/10.1371/journal.pone.0112821 (2014).

Liben-Nowell, D. and J. Kleinberg, “The link-prediction problem for social networks”, J. Am. Soc. Inf. Sci. Technol. pp. 1019–1031, URL http://dx.doi.org/10.1002/asi.v58:7 (2007).

Newman, M. E. J., “Spectral methods for network community detection and graph partitioning”, CoRR (2013).

Newman, M. E. J. and M. Girvan, “Finding and evaluating community structure in networks”, Physical Review , 026113 (2004).

Newman, M. J., “A measure of betweenness centrality based on random walks”, Social Networks URL http://www.sciencedirect.com/science/article/pii/S0378873304000681 (2005).

Page, L., S. Brin, R. Motwani and T. Winograd, “The pagerank citation ranking: Bringing order to the web”, (1998).

Pons, P. and M. Latapy, “Computing communities in large networks using random walks”, in “Proceedings of the 20th International Conference on Computer and Information Sciences”, ISCIS’05, pp. 284–293 (Springer-Verlag, Berlin, Heidelberg, 2005), URL http://dx.doi.org/10.1007/11569596_31.

Ruppert, E., “Finding the k shortest paths in parallel”, (2012).

Singh, A. P. and D. P. Singh, “Implementation of k-shortest path algorithm in gpu using cuda”, Procedia Computer Science URL http://www.sciencedirect.com/science/article/pii/S1877050915006122, international Conference on Computer, Communication and Convergence (ICCC 2015) (2015).

Spielman, D. A. and S.-H. Teng, "A local clustering algorithm for massive graphs and its application to nearly linear time graph partitioning", SIAM Journal on Computing URL https://doi.org/10.1137/080744888 (2013).

Staudt, C. L., Y. Marrakchi and H. Meyerhenke, "Detecting communities around seed nodes in complex networks", in "2014 IEEE International Conference on Big Data (Big Data)", pp. 62–69 (2014).

Takaffoli, M., R. Rabbany and O. R. Zaane, "Community evolution prediction in dynamic social networks", in "2014 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2014)", pp. 9–16 (2014).

Tantipathananandh, C., T. Berger-Wolf and D. Kempe, "A framework for community identification in dynamic social networks", in "Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining", KDD '07, pp. 717–726 (ACM, New York, NY, USA, 2007), URL http://doi.acm.org/10.1145/1281192.1281269.

Tong, H., J. He, M. Li, W.-Y. Ma, H.-J. Zhang and C. Zhang, "Manifold-ranking-based keyword propagation for image retrieval", EURASIP Journal on Applied Signal Processing (2006).

von Luxburg, U., "A tutorial on spectral clustering", Statistics and Computing URL https://doi.org/10.1007/s11222-007-9033-z (2007).

White, S. and P. Smyth, *A Spectral Clustering Approach To Finding Communities in Graphs*, pp. 274–285 (2005), URL http://epubs.siam.org/doi/abs/10.1137/1.9781611972757.25.

Yin, H., A. R. Benson, J. Leskovec and D. F. Gleich, "Local higher-order graph clustering", in "Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining", KDD '17, pp. 555–564 (ACM, New York, NY, USA, 2017), URL http://doi.acm.org/10.1145/3097983.3098069.

Yong-Yeol Ahn, S. L., James P. Bagrow, "Link communities reveal multiscale complexity in networks", Nature pp. 761–764 (2010).

Zhang, J., J. Tang, C. Ma, H. Tong, Y. Jing, J. Li, W. Luyten and M.-F. Moens, "Fast and flexible top-k similarity search on large networks", ACM Trans. Inf. Syst. (2017).

Zhao Yang, C. J. T., Ren Algesheimer, "A comparative analysis of community detection algorithms on artificial networks", Scientific Reports (2016).

Zhou, D., S. Zhang, M. Y. Yildirim, S. Alcorn, H. Tong, H. Davulcu and J. He, "A local algorithm for structure-preserving graph cut", in "Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining", KDD '17, pp. 655–664 (ACM, New York, NY, USA, 2017), URL http://doi.acm.org/10.1145/3097983.3098015.

Zhukov, L., “Structural analysis and visualization of networks”, URL https://www.youtube.com/watch?v=jIS5pZ8doH8&list=PLriUvS7IljvkBLqU4nPOZtAkp7rgpxjg1&index=11 (2015).