Mining Connection Pathways for Marked Nodes in Large Graphs

Leman Akoglu
SUNY at Stony Brook
leman@cs.stonybrook.edu

Jilles Vreeken
University of Antwerp
jilles.vreeken@ua.ac.be

Hanghang Tong
City College, City University of NY
tong@cs.ccny.cuny.edu

Duen Horng Chau
Georgia Tech
polo@gatech.edu

Nikolaj Tatti
K.U. Leuven
nikolaj.tatti@cs.kuleuven.be

Christos Faloutsos
Carnegie Mellon University
christos@cs.cmu.edu

Abstract

Suppose we are given a large graph in which, by some external process, a handful of nodes are marked. What can we say about these nodes? Are they close together in the graph? or, if segregated, how many groups do they form? We approach this problem by trying to find sets of simple connection pathways between sets of marked nodes.

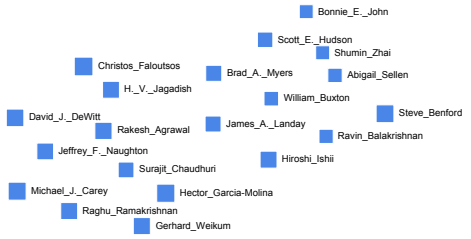
We formalize the problem in terms of the Minimum Description Length principle: a pathway is simple when we need only few bits to tell which edges to follow, such that we visit all nodes in a group. Then, the best partitioning is the one that requires the least number of bits to describe the paths that visit all the marked nodes.

We prove that solving this problem is NP-hard, and introduce DOT2DOT, an efficient algorithm for partitioning marked nodes by finding simple pathways between nodes. Experimentation shows that DOT2DOT correctly groups nodes for which good connection paths can be constructed, while separating distant nodes.

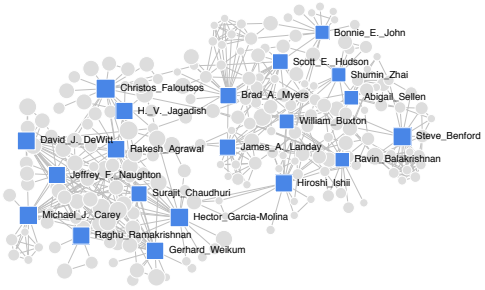
1 Introduction

Suppose we are given a large graph $G = (V, E)$ in which, by some external process, a handful of nodes $M \subseteq V$ are marked. How can we explain the relations among these marked nodes? Do they 'talk to' each other, and thus are 'close-by'? Or, are they 'unrelated', or 'far away', to each other? More formally, the key question we address is: How can we use the network structure to explain a given set of marked nodes, by partitioning them such that there are simple paths connecting the nodes in each group, while nodes in different parts are not easily reachable?

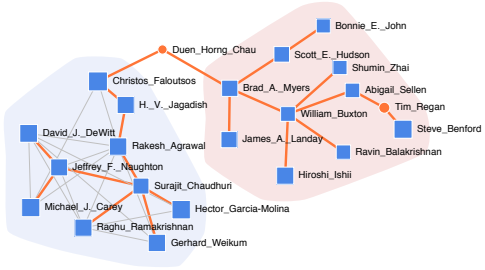
For example, consider Figure 1. In (a), a list of 20 authors from DBLP are marked. In this plot, there is no information (other than author names) that explains any correlation among the authors. In (b), the marked nodes are projected to and highlighted in the co-authorship graph. In contrast, here it is hard to observe any patterns as there is information overload. Our result is shown as (c), which explains the marked nodes with two well-separated groups, revealing simple connections and connectors among all the marked nodes.



(a) What to say about this "list" of authors?



(b) Any patterns? "Too many" connections.



(c) The "right" connections → Better sensemaking

Figure 1: 20 chosen researchers from DBLP (blue squares). Edges denote co-authorship. DOT2DOT finds simple connection pathways revealing two groups: VLDB and CHI.

To the best of our knowledge, this is a novel problem that has not been studied before—though it has many use cases. In this paper, we formally define and formulate the problem using information and graph theoretic principles and propose effective solutions. While the problem is theoretically interesting in its own right, it also has several motivating applications. We highlight two examples below, and refer to [1] for more applications.

- Given a dynamic event over a network (e.g., people affected by a certain disease or buying a particular product), how can we group the nodes such that the network structure can be associated with the spread of the event within groups but not quite across groups?
- Given the Web graph and a set of top ranked pages (nodes) returned by some keyword search like Google, how can we group these pages into groups and reveal connection pathways among them, rather than simply listing them ignoring their relations on the Web?

Intuitively, we want to partition a given set of marked nodes such that the nodes within a part are 'close-by' while nodes across parts are far apart. In addition, for the 'close-by' nodes in each part, we want to find a 'succinct' subgraph connecting them. Moreover, we rather not visit nodes of very high degree, such as hubs in social networks, because those nodes connect to virtually everything in the graph, and they do not provide much information by association. Therefore, we think of such high degree nodes as separators, and try to avoid including them in our 'succinct' subgraphs.

We formalize this problem in terms of the Minimum Description Length principle [23]: a collection of paths is simple when we need few bits to direct the user from one node to the other. Hence we typically do not want to visit nodes of high degree, as it is more expensive to identify which edge to follow leading from it. Similarly, we require more bits if we have to visit many unmarked nodes in order to arrive to the next marked node. As such, the best 'explanation', is the one for which we need the least number of bits to identify all marked nodes.

We show this problem is NP-hard, and has connections to well-known other problems in graph theory. We discuss a number of fast methods for finding a partitioning and its respective simple paths, and introduce DOT2DOT, an efficient algorithm for explaining marked nodes in large graphs. Experimentation shows that DOT2DOT correctly groups nodes for which simple paths can be constructed, while separating distant nodes.

As we mentioned above, explaining marked nodes in a graph is a novel problem that has not been explored before. There, however, exist relations to existing proposals. Graph clustering [9, 10, 15], for instance, is related in that we aim at grouping nodes that are nearby—yet our goal is not to cluster the full graph, but only the marked nodes, yet making use of the graph structure. Other related methods include connection subgraphs [8], center-piece subgraphs [28], local graph clustering [3], and seed set expansion [22]. Each of these proposals, however, targets a different problem, and does not provide sensemaking via partitioning. Section 5 includes more detailed discussion on related work.

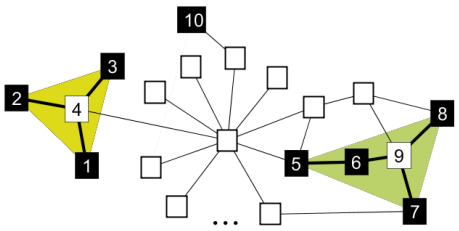


Figure 2: Toy graph with 8 marked (black) nodes. Our DOT2DOT algorithm automatically 'describes' them in 3 groups, discovering 'missing connectors' (nodes '4' and '9').

Our major contributions are as follows, 1) we formally define the problem of 'describing marked nodes in graphs', 2) we formulate it in terms of information theoretic principles, 3) we propose fast methods for finding good partitions and high quality connection pathways, and 4) we experimentally evaluate our methods on real and synthetic data.

2 Formulating Paths: Theory

2.1 Problem Definition Given a graph $G = (V, E)$ and a set of marked nodes $M \subseteq V$, we consider the following two related problems.

Problem 1. Optimal partitioning Find a coherent partitioning P of M. Find the optimal number of partitions $|P|$.

Problem 2. Optimal connection subgraphs Find the minimum cost set of subgraphs connecting the nodes in each part $p_i \in P$ efficiently.

The two problems we consider above involve three sub-problems: **(sub-problem 1)** how to define the 'coherence' of a set of nodes, **(sub-problem 2)** how to define the 'cost' for a connection subgraph, and **(sub-problem 3)** how to find the connection subgraph(s) quickly in large graphs.

As an example, consider Figure 2. We depict a toy graph in which 8 nodes have been marked. It is clear to see from the figure that the marked nodes naturally form three groups: $p_1 = \{1, 2, 3\}$, $p_2 = \{10\}$, and $p_3 = \{5, 6, 7, 8\}$, all well separated by the big star-node in the middle. Although only marked nodes 5 and 6 are connected, for each part p_i there exists a simple dot-to-dot 'road map' we can follow, in order to visit all marked nodes in that part—without having to visit too many unmarked nodes, such as node 4 in p_1.

As we aim to describe the given marked nodes as succinctly as possible, we use the Minimum Description Length (MDL) principle from Information Theory [23].

2.2 Objective Function The key idea behind our method builds on an encoding scheme, involving one sender and one receiver. We assume both the sender and receiver know graph structure $G = (V, E)$ and only the sender knows the set of marked nodes M. The goal of the sender, then, is to come up with an encoding to *transmit to the receiver the info. of which nodes are marked, using as few bits as possible*.

One straightforward way to encode the set of marked nodes is to encode their node-id's separately, using $\log |V|$ bits each. On the other hand, it might be more efficient to exploit the neighborhood information for 'close-by' nodes. In the simplest case, for example, if nodes u and v are both marked and they are direct neighbors, i.e. $e(u, v) \in E$, then the sender can encode u's id and then use only $\log d(u)$ bits to encode v, where we assume a canonical order on nodes (say by increasing node-id) and $d(u)$ denotes the degree of u in G. As such, the sender follows a path from one marked node to the other to encode 'close-by' nodes. Depending on the graph structure, from time to time it might be more efficient to restart encoding from a new node by directly providing its id, in case the cost of the path to that node exceeds $\log |V|$ bits. In fact, that is exactly what determines the partitioning P of the nodes.

Simply put, one can imagine the way of encoding as hopping from node to node for encoding close-by nodes and from time to time flying to a completely new node for encoding farther nodes until all marked nodes are encoded. This resembles a tour on the graph that travels from a marked node (or, dot) to another, so succinctly describing the marked set. Hence, the name DOT2DOT.

In effect, we are after the shortest description for a group of marked nodes $M \subseteq V$ in a graph $G = (V, E)$. More generally, the idea is that per part p_i of P, we find the easiest/simplest subgraph T in G that spans *at least* all marked nodes in p_i. Simplicity of T is determined by the number of nodes we visit in this tour, how many unmarked nodes we visit, and in particular how easily per visited node we can identify which edge we have to follow next; nodes with (very) high degree hence make the path more complex, or, less likely. Also notice that the simplest subgraph would in fact be a ***tree*** since it would require less bits to refer to a node we have already visited in our encoding.

In this section, we describe the cost function for a given partitioning P and the given connection trees for each part p_i. We use this cost function as our objective function that we aim to minimize for model selection.

More formally, we first transmit the number of partitions $|P|$, for we need at most $\log |V|$ bits—as there will be at most $|V|$ marked nodes in total, which in the worst case are all put in separate parts—so we have $L(|P|) = \log |V|$.

Then, per part $p_i \in P$, we have a tree T spanning at least the marked nodes of p_i. To identify the root node of T in G we have to spend $\log |V|$ bits.

Then, recursively per node $t \in T$, we transmit how many branches t has, denoted by $|t|$. As t corresponds to a node v_t in G, and $d(v_t)$ gives us the degree of v_t, we can transmit $|t| \leq d(v_t)$ in $\log d(v_t)$ bits. On the other hand, since a simple tree would presumably have small branch-out factor, we choose to encode $|t|$ using universal integer encoding [11]. This encoding specifies that in order to encode a non-zero positive integer n we require $L_{\mathbb{N}}(n) = \log^\star n + \log(c)$ bits with $c = \sum 2^{-\log n} \approx 2.865064$, and $\log^\star(n) = \log n + \log\log n + \cdots$ sums over all positive terms. So, as $|t|$ can be zero (for leaves), we transmit its value in $L_{\mathbb{N}}(|t| + 1)$ bits.

Next, we identify which out-edges of v_t have to be visited. This we can encode most succinctly by assuming a canonical order of all possible subsets of selected edges of that size, and transmitting the index of the actual subset. This takes $\log \binom{d(v_t)}{|t|}$ bits.

Leaf-nodes are easily identified as their number of branches is given as 0. If such is the case, we traverse back up the tree until we find a node with an unvisited branch. Once all branches have been visited, we stop transmitting the structure of the tree.

Now that we know which nodes $p_i \subset V$ are in our tour, we need to know which of these are marked. Let $|T|$ denote the number of nodes in T, and $||T||$ the number of marked nodes in T. As the recipient now knows the tree, and hence $|T|$, and $||T|| \leq |T|$ we need $\log |T|$ bits to transmit $||T||$. Next, we need to identify which $||T||$ nodes of T are marked, which we again do by a binomial: $\log \binom{|T|}{||T||}$.

As such, for one part $p_i \in P$, we have

$$L(p_i) = \log |V| + L(t) + \log |T| + \log \binom{|T|}{||T||}$$

in which $L(t)$ for a node t in p_i is defined recursively as

$$L(t) = L_{\mathbb{N}}(|t| + 1) + \log \binom{d(v_t)}{|t|} + \sum_{j}^{|t|} L(b(t, j)) \quad ,$$

where $b(t, j)$ identifies the node t' we reach from node t by descending branch j. Notice that by being recursive, $L(t)$ encodes the branching cost for all tree-nodes. Putting this together, we get as the total encoded size for a path P, marking a group of nodes M, in a graph G

$$L(P, M \mid G) = L(|P|) + \sum_i L(p_i) \quad .$$

Note that our initial assumption that both the sender and the receiver know G does not affect model selection: as G is constant for all possible sets of marked nodes on G, explicitly transmitting G would only add a constant cost for all possible models under consideration, and hence not influence measuring the quality of a model.

Given the above formalization, we now only have to find the optimal partitioning P of M, and the optimal tours per part $p_i \in P$ such that $L(P, M \mid G)$ is minimized.

2.3 Problem Formulation and NP-hardness The total encoding cost $L(P, M \mid G)$ can score a given set of solutions and point to the best among them, however it does not provide direct means to find the optimal solution.

In this section, we show that this problem is NP-hard, with a reduction to the Steiner tree problem: given an undirected unweighted graph $G = (V, E)$ and a subset of nodes $X \subseteq V$, the objective is to find the minimum cost tree that spans all the nodes in X. The cost of a tree is defined as the total number of nodes, i.e. total number of edges plus one, in the tree. Note that the tree may include nodes in X, as well as other nodes which are typically referred to as Steiner nodes. Steiner trees are well-known in graph theory, and find application in multicast models for finding good server nodes in computer networks for avoiding network congestion and reducing latency in multi-user streaming data settings.

THEOREM 2.1. *Minimizing $L(P, M \mid G)$ is NP-hard.*

Proof. See [1].

3 Finding Good Paths: Methods

Since minimizing $L(P, M \mid G)$ is NP-hard, we are interested in fast approximations. To find fast solutions, we will exploit heuristics for the directed Steiner (d-Steiner) tree problem [5], which is a well-studied combinatorial optimization problem, for which algorithms that provide approximation guarantees have been proposed [5, 12, 20, 32]. However, while providing fairly good bounds, these algorithms all require great amounts of computing power, making them intractable for application on large graphs. In this section we therefore propose fast heuristics for large graphs.

Before we proceed with proposed solutions, we first define how we transform the input graph G to a directed weighted graph G', which we use in obtaining a solution.

DEFINITION 1. *Given an undirected unweighted graph $G = (V, E)$ and a set of marked nodes $M \subseteq V$, the transformed graph $G' = (V, E')$ is a directed weighted graph in which each edge $e(u, v) \in E$ is replaced by two directed edges $e'(u, v) \in E'$ and $e'(v, u) \in E'$, for which the weights $w'(u, v) = \log d(u)$ and $w'(v, u) = \log d(v)$.*

3.1 Proposed Algorithms Given the transformed directed weighted graph G', we want to find *the set of trees with minimum total cost* on the marked nodes. Without loss of generality we assume that the marked nodes will be the leaf nodes in the resulting trees (if a marked node is a non-leaf node, we can always add its copy and connect it to its copy with a zero-cost edge), therefore from hereafter we refer to the marked nodes as the *terminals*.

Procedure 1: `FindBoundedPaths` (G, T)

Input: A graph $G = (V, E)$, terminals $T \subseteq V$
Output: Asc. length short paths SP from terminals

1 $lengths \leftarrow \log(degree(V))$, $SP \leftarrow \emptyset$
2 **foreach** $t \in T$ **do**
3 $paths \leftarrow \emptyset$, $pathcosts \leftarrow \emptyset$, curlen $\leftarrow 0$
4 $paths.add(\{t\})$, $pathcosts.add(lengths(t))$
5 curlen $\leftarrow$ curlen + $\min(pathcosts)$
6 **while** curlen $< \log(|V|)$ **do**
7 $pathcosts \leftarrow pathcosts - \min(pathcosts)$
8 **foreach** v *s.t.* $pathcosts(v) = 0$ **do**
9 $path \leftarrow paths(v)$
10 **foreach** $n \in N_v$ **do**
11 **if** $n \notin path$ **then**
12 SP.add(t, n, curlen, $path$)
13 $paths$.add($\{path, n\}$)
14 $pathcosts$.add($lengths(n)$)
15 $paths$.remove(v), $pathcosts$.remove(v)
16 curlen $\leftarrow$ curlen $+ \min(pathcosts)$
17 **return** SP

3.1.1 Finding Bounded-length Paths Most of our proposed methods use short paths between the terminals as a starting point. Hence, we first present an algorithm, `FindBoundedPaths`, to find *multiple* short paths up to a certain length (hence, indirectly, up to a certain cost), in order of increasing length between the terminal nodes. The threshold on the length of the paths is not a parameter; as it takes $\log |V|$ bits to start a new partition, we set it to $\log |V|$.

`FindBoundedPaths` employs a BFS-like expansion starting from each terminal until the threshold path length is reached. The paths from the terminal to the nodes encountered over this expansion as well as their total lengths are stored. A major advantage of our transformed graph formulation for the BFS-like expansion is that the cost of all out-edges for a node v are the same and equal to $\log d(v)$. As a result we only need to keep the length per node, rather than per edge, in our BFS-list.

The pseudo-code is given in Procedure 1. *lengths* is a vector of size $|V|$ that holds $\log d(v)$ for each node $v \in V$, and SP is a structure list that stores the paths from a terminal to other nodes encountered in the BFS expansion (line 1). The paths are indexed by their end nodes and stored in increasing order of their lengths. *paths* is a dynamic list that stores all current list of paths during the BFS expansion, and *pathcosts* stores their respective lengths (3).

In each iteration, we expand the node(s) with the minimum length (7). For each such node v (8), we expand to its neighbors N_v (10). We first store the path information to these neighbors (12), and then add the new paths from

the root to these neighbors to the *paths* list (13), and their respective lengths to the *pathcosts* list (14) for further expansion. Note that a node can appear multiple times in our BFS lists since each expansion is associated with only a single unique path (that is, the path from the root in the BFS tree), and a node may belong to multiple paths. We continue iterating, i.e. expanding nodes with minimum length (16) until the threshold length is reached (6).

At the end of `FindBoundedPaths`, we have all the paths from every terminal to all the nodes in G' that are within a length $\log|V|$ path, ordered in increasing length. Notice that the expansion can be performed completely in parallel for each terminal for speed.

Next, we present our fast approximate solutions for finding a low-cost set of trees on the terminals.

3.1.2 DOT2DOT-ConnectedComponents A simple solution to partition the terminals is to consider the connected components they induce on the graph. That is, the terminals that are directly connected are put in the same and otherwise separate parts. By definition, all the edges are reciprocated in the transformed graph, and therefore the subgraph induced on each part including two or more nodes is not a directed acyclic tree (it may as well have cycles of length greater than 2). To find the minimum cost rooted directed tree(s) (a.k.a. arborescence), we use the Chu-Liu algorithm [7].

3.1.3 DOT2DOT-MinArborescence Our second method uses the *transitive closure* graph $G_t = (T, E_t)$ of the terminals to find a minimum arborescence. G_t consists of the terminals and directed edges $e(t_i, t_j) \in E_t$ between them having weight equal to the shortest path length $w(t_i \rightsquigarrow t_j)$ from t_i to t_j, $1 \leq i, j \leq |T|$. If the shortest paths between all pairs of terminals are of length less than $\log|V|$, then G_t is simply a directed clique graph. As we find up to length $\log|V|$ paths from every terminal in `FindBoundedPaths`, a terminal does not have an edge in G_t to those terminals that are more than this threshold apart from it.

Having constructed the transitive closure graph, we add a so-called universal node u with directed edges $e(u, t_i)$ to every terminal t_i with weight $\log|V|$. We find the minimum weight arborescence A in this new graph. Since the universal node u does not have any incoming edges, it constitutes the root of A. In fact, the number of out-edges of u in A gives us the number of parts $|P|$, and its sub-trees constitute the partitioning P. Next, we replace the edges in each of u's sub-trees with their corresponding paths in the transformed graph G'. The expanded sub-trees may contain both marked and unmarked nodes. Also notice that the expanded sub-trees might no longer be trees but contain cycles. Therefore, we rerun the arborescence algorithm on the expanded sub-trees and remove any unmarked leaf nodes in the resulting arborescences, which yields the final forest of Steiner trees. The min-arborescence algorithm takes $O(|V|^2)$ for dense graphs. As we run it on the closure of marked nodes only, we get $O(|M|^2)$. The detailed pseudo-code is given in [1].

3.1.4 DOT2DOT-1-Level Tree Our third method builds a (set of) level-1 tree(s) from the transitive closure graph on the terminals. Simply put, we try each terminal as the root and connect it to the other terminals with shortest paths on the transformed graph G'. If the selected root does not have shorter than length $\log|V|$ paths to all the terminals, a (set of) level-1 tree(s) is built on the remaining terminals. We return the tree(s) with the minimal total encoded length as the forest of Steiner trees. Note that this heuristic algorithm provides a $|M|$-approximation to the optimal solution [5]. The detailed pseudo-code is given in [1].

3.1.5 DOT2DOT-k-Level Tree Our final method builds on and generalizes the 1-level tree heuristic to k-level trees. The goal is to start with a (set of) 1-level tree(s) and successively refine each for lower cost. The main idea is the following: for a given tree with root r, find one or more intermediate nodes $v \in V$, such that the total cost from r to each v plus the costs of sub-trees rooted at v's (each with a mutual set of terminals as leaves) reduces the initial cost.

More specifically, we construct a k-level tree by a union of sub-trees, each consisting of the root r, exactly one intermediate node v, and all the descendants of this intermediate node. We refer to such sub-trees of level k as *partial k-tree*s, and we find them in a greedy manner described as follows. First, we find a *partial k-tree* L_k^p (if any) that reduces the cost for a subset of terminals that it spans. Next, we remove the terminals spanned by L_k^p and iterate this process until all terminals are spanned. Algorithm 1 formally describes the k-level tree heuristic.

To complete the k-level heuristic, we need to describe its subroutine `Partial`k`Tree` which, given a root r and a set of terminals S, finds a low-cost partial k-level tree rooted at r and spanning (a subset of) the terminals. The `Partial`k`Tree` heuristic is recursive: in order to find a partial k-level tree, we need to first find certain partial $(k$-1)-level trees that span all the given terminals. The base case is reached for level $k = 2$, which works as the following. For each candidate v for the intermediate node, we sort the terminals S according to the potential savings of inserting node v between the root r and each terminal in S. The potential saving for a terminal t_i stands for the difference between the shortest path lengths $w(r \rightsquigarrow t_i)$ and $w(v \rightsquigarrow t_i)$. These savings can take positive and negative values and are sorted in decreasing order. Finally, we include consecutive terminals from this sorted list while their inclusion decreases cost of the *partial 2-tree*.

The detailed pseudo-code of `Partial`k`Tree` and further details on the k-level tree algorithm can be found in [1].

Algorithm 1: DOT2DOT-k-LEVELTREE

Input: A graph $G = (V, E)$, terminals $T \subseteq V$
Output: Partitions P: a Steiner tree on each $p \in P$

```
SP ← FindBoundedPaths (G, T)
Build candidate-graph G_c=(V_c, E_c): union of all top
  3 shortest paths between all pairs of terminals.
mincost ← ∞
foreach t ∈ T do
    L_k ← ∅
    L_1 ← min-cost 1-level tree(s), one rooted at t
    foreach L_1 ∈ L_1 do
        S ← leaves(L_1), r ← root(L_1)
        if |S| < 2 then continue
        while S ≠ ∅ do
            L_k^p ← PartialkTree(G_c, SP, r, S, k)
            L_k ← L_k ∪ L_k^p
            S ← S \ leaves(L_k^p)
    P ← ExpandPartition (L_k, SP, T)
    cost ← L(P, M | G)
    if cost < mincost then
        mincost ← cost, minP ← P
return minP
```

Name	$\|V\|$	$\|E\|$	Description
Netscience	379	914	Author collaborations
GScholar	83K	148K	Academic citations
DBLP	329K	1094K	Author collaborations

Table 1: Dataset summary.

4 Empirical Study

In this section, we evaluate our proposed method; first we give intuitive results on synthetic examples, second we quantitatively compare the performance of the four proposed heuristic methods DOT2DOT-* (COMPONENTS, MINARBORESCENCE, 1-, and 2-LEVELTREE), and finally we provide case studies to show qualitative performance on real-world graphs. We provide our DOT2DOT as a standalone tool[1] for research purposes, and our datasets are all publicly available (see Table 1).

4.1 Synthetic examples We start by testing our method through three examples on a synthetic 100×100 grid graph, as well as one example on the well-known Zackary's karate club network [31].

As shown in Figure 3(a), we select 8 marked (blue square) nodes relatively close to each other on the grid graph, and run all four of our approximation methods. We highlight (by orange bold edges) the minimum description cost tree found (orange nodes: connectors)—notice that it provides succinct connections among the marked nodes. When we place 4 of the marked nodes farther apart in the grid graph, our method successfully partitions the marked nodes and provides 2 connection trees for each as shown in Figure 3(b).

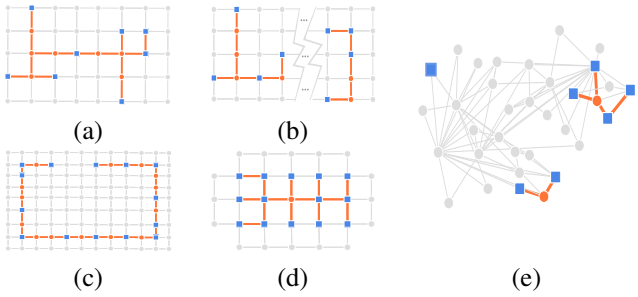


Figure 3: Examples: (a) 8 marked (square) nodes placed close to each other on a grid, (b) 8 nodes placed apart on the grid, forming 2 parts, (c) connecting the dots, (d) recovering missing connector, (e) 7 marked nodes on Karate graph.

Next, we place the marked nodes intermittently on the grid in Figure 3(c). Intuitively, human would connect these dots to form a rectangle—and so does our method. Figure 3(d) shows a set of marked nodes forming an almost full rectangle on the grid, except one node in the middle left unmarked. Notice that our method successfully recovers this 'left-behind' node as a significant connector.

Finally, DOT2DOT partitions 7 marked nodes on the Karate graph as depicted in Figure 3(e) into 3 parts; in which parts are well separated through high degree hub nodes.

4.2 Comparing the algorithms In this section, we aim to understand the average performance of the four approximation methods proposed in § 3. To do so, we run simulations on three real-world graphs and compare their average description cost. In each simulated run we test the algorithms on a different set of marked nodes. We first describe the datasets we used and then provide details on the simulations.

The dataset information is given in Table 1. *Netscience*[2] is a co-authorship network of scientists working on network theory and experiment. *GScholar*[3] contains academic articles published in venues of various fields of computer science and their citation relations. *DBLP*[4] is also a co-authorship network of researchers in computer science. All the datasets are publicly available.

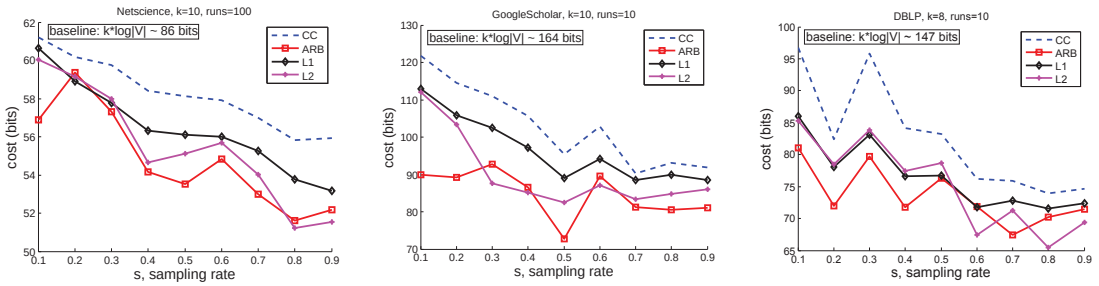
For each simulation the set of marked nodes is selected

[1] Source code of DOT2DOT: http://www.cs.stonybrook.edu/~leman/pubs.html#code

[2] http://www-personal.umich.edu/~mejn/netdata/

[3] http://dl.dropbox.com/u/17337370/gscholar.db

[4] dblp.uni-trier.de/xml/



(a) *Netscience* (b) *GScholar* (c) *DBLP*

Figure 4: Comparison of the methods: total cost (bits) versus various sampling rates s to choose the marked nodes. The marked nodes are 'closer' in the graph for larger s—hence smaller expected cost. Among heuristics, COMPONENTS (CC) is costliest and the best MINARBORESCENCE (ARB) achieves up to 49% savings over baseline on average.

via a variation of snowball sampling, which works as follows. To select a set of k marked nodes, we follow a random walk sampling scheme. First, we fix a sampling rate s. Then, we choose and mark a node at random in the given graph. Next we randomly visit $k' < k$ of its neighbors, and mark each visited neighbor with probability s. We continue this process from a random node already visited (either marked or unmarked) until we have k marked nodes in total.

In Figure 4, we show the average description cost (in bits) versus sampling rate $s = \{0.1, \ldots, 0.9\}$ of our simulations (k = 8-10, k' is set to 3, $k' = \{1, 2\}$ gives similar results). The results are averaged over 100 runs for the smaller *Netscience*, and 10 runs for the rest. We notice comparable performance results on all three graphs; the simplest heuristic COMPONENTS provides the costliest, while MINARBORESCENCE gives the most succinct description among the four methods on average. In addition, 2-LEVELTREE provides competitive results to MINARBORESCENCE, and outperforms 1-LEVELTREE as would be expected.

In addition, notice the downward trend of the cost for increasing sampling rate. This is expected, as for higher s the marked nodes are chosen among closer ones. This also explains the relatively larger gap in performance between 2-LEVELTREE and MINARBORESCENCE for small s, as for farther apart nodes a higher level tree may be required.

Note that by MDL, we only care about the relative cost of the methods rather than their actual magnitude of cost. The reason is that compression itself is not our goal, but a means to identify the best model for the data. For example, we find that the best performing method MINARBORESCENCE required 36%, 48%, and 49% fewer bits on average for *Netscience*, *GScholar*, and *DBLP*, respectively than the baseline $k \log |V|$ (encoding marked nodes directly). On the other hand, since the number of marked nodes is often small, the baseline cost is not drastic—less than 200 bits for our graphs (see Figure 4). However while it has somewhat small and comparable score, it provides *no* information about the relations among marked nodes.

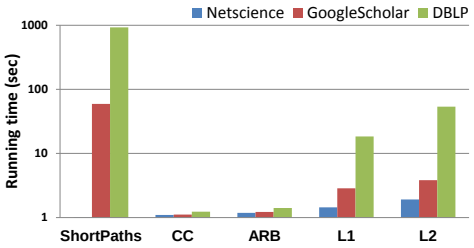


Figure 5: Average run time of proposed methods.

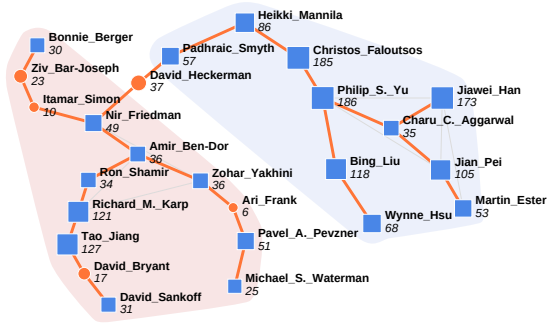
Finally, we give the running time of our methods in Figure 5. We notice that due to the iterative nature of looking for good intermediate nodes, k-LEVELTREE heuristics (L1 and L2 in the figure) take longer than the others. At the same time, L2 completes in about 50 seconds on our largest graph *DBLP* and can be further sped up by providing it with a smaller candidate graph. Since `FindBoundedPaths` takes the most considerable time in our framework, we propose to run all heuristics and report the best result with the minimum cost among them.

4.3 Case studies on real graphs

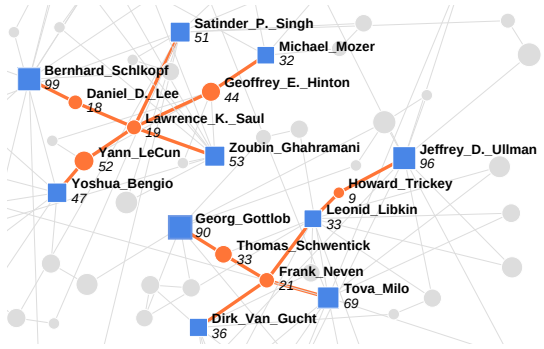
Our method can provide useful insight about the connections and connectors among a group of nodes in a given graph. As such, we develop DOT2DOT as an interactive tool that aids in visualization and sensemaking. In this section, we provide qualitative analysis on two large real-world graphs.

Authors in *DBLP* We first employ DOT2DOT on the well-known *DBLP*citation graph. In particular, we select 2 major conferences in certain fields and mark the top 10 authors from each, who have the most number of papers appearing at that particular conference.

In Figure 6(a) we show the connection tree among researchers from RECOMB (computational biology) and KDD (data mining and machine learning). Notice the simple connections among the authors within the same field and



(a) *DBLP*: RECOMB vs. KDD



(b) *DBLP*: NIPS vs. PODS

Figure 6: Connection trees among researchers from *DBLP* with most number of papers at specified conferences. (a) connector David Heckerman: mining biomedical data, (b) two trees (researcher groups) sufficiently apart.

otherwise well separated communities. The connector is David Heckerman, the director of the eScience team at Microsoft Research whose work focuses on foundations of learning and its applications on biological and medical data.

In the next example from *DBLP* we look at authors from NIPS (machine learning) and PODS (database systems), 5 authors from each. DOT2DOT connects the authors from each field through a few connectors as shown in Figure 6(b). Notice that there are two parts in our partitioning; which suggests that the authors from these two communities are sufficiently apart in the graph.

Articles in *GScholar* We also apply DOT2DOT to summarize academic articles on certain topics from *GScholar*. By their titles, we mark nodes in the citation graph containing specified keywords.

In Figure 7 we show 8 marked articles containing both 'large graphs' and 'visual' keywords in their title. Our visualization highlights the resulting partitioning on the candidate graph (recall that candidate graph contains the union of top-3 shortest paths among all pairs of marked nodes). We find that DOT2DOT partitions the marked nodes into 4 parts, 3 of which are singletons. The connection tree for the

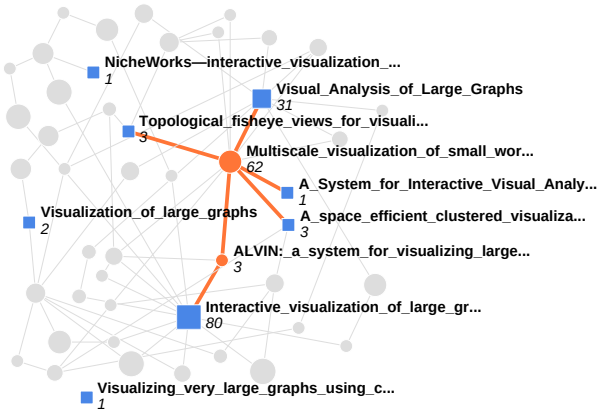


Figure 7: Connection trees among articles from *GScholar* with keywords 'large graphs' and 'visual': one tree summarizing 5 marked nodes while singletons reside farther apart.

largest part provides a concise summary for the 5 nodes in this part, with only two connectors. Also notice that the singleton nodes are at least 3 hops apart from this tree, which is evident looking at the candidate graph.

5 Related Work

In graph clustering [9, 10, 15], although related, the goal is to cluster the *whole* graph—we try to partition and connect a small *subset* of nodes, making use of the graph structure.

One closely related line of work is the connection subgraphs mining problem [8]. Later, [21, 24] extended [8] for querying labeled graphs in which nodes and edges belong to certain types or relations. On similar lines, [13] and [25] propose techniques to discover a coherent chain of evidence trails that connect two given concepts in text documents. All these methods however, work only for pairs of nodes.

The center-piece subgraph problem [28, 29] finds the most 'centered' node that has strong connections to most or all of the query nodes. Koren et al. [16] extracts the subgraph with at most a certain size which retains most of the proximity between query nodes. Connection subgraphs are also exploited for graph visualization and summarization [6, 14]. In turn, [2, 3, 22, 27] focus on expanding communities around a given set of seed nodes. [4] develops methods to find effective connection trees among keywords for browsing and querying. Leskovec et al. [17] use features extracted from query connection subgraphs together with ML techniques to predict quality of query results. [30] finds the top-k nodes with the highest 'gateway-ness' score with respect to a given source and target node set, such that they collectively lie on most of the short paths from source nodes to target nodes.

Our work differs from existing work in two major aspects; (1) we do not assume any specific connection structure among the query nodes (e.g., source-sink [8], star-

shape [28], etc.); and (2) we employ information theoretic ideas to find the optimal partitioning and connections in a principled way, without any user-defined parameters.

6 Concluding Remarks

We propose a novel problem—how to explain a set of marked nodes using the connectivity structure of the graph. Intuitively, the goal is to find groups of 'close-by' nodes, together with 'simple' connections among them. To the best of our knowledge, we are the first to give formal problem definitions and initial solutions.

Our contributions are: (1) *Problem formulation:* We formalize the problem of 'describing a set of chosen nodes in a graph' in terms of the Minimum Description Length principle: the best description is the one for which we need the least number of bits to encode the marked nodes; (2) *Fast algorithms:* We show this problem has connections to directed Steiner trees [5], and prove that finding the optimal solution is NP-hard. We propose fast solutions for large graphs; (3) *Experiments* on synthetic and real collaboration and citation graphs demonstrate the effectiveness of DOT2DOT in discovering good connectors and connections.

Acknowledgements

We thank the anonymous reviewers for their useful feedback. Research was sponsored by NSF under Grant No. IIS1017415, ARL under Coop. Agreement No. W911NF-09-2-0053, and ADAMS program sponsored by DARPA under Agreements No.s W911NF-12-C-0028, W911NF-11-C-0200, and W911NF-11-C-0088. Jilles Vreeken is supported by a Post-Doctoral Fellowship of the Research Foundation – Flanders (FWO).

References

[1] L. Akoglu, J. Vreeken, H. Tong, D. H. Chau, N. Tatti, and C. Faloutsos. Islands and bridges: Making sense of marked nodes in large graphs. (CMU-CS-12-124), August 2012.

[2] R. Andersen, F. R. K. Chung, and K. J. Lang. Local graph partitioning using pagerank vectors. In *FOCS*, 2006.

[3] R. Andersen and K. J. Lang. Communities from seed sets. In *WWW*, pages 223–232, 2006.

[4] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using banks. In *ICDE*, pages 431–440, 2002.

[5] M. Charikar, C. Chekuri, T.-Y. Cheung, Z. Dai, A. Goel, S. Guha, and M. Li. Approximation algorithms for directed steiner problems. In *SODA*, pages 192–200, 1998.

[6] D. H. Chau, C. Faloutsos, H. Tong, J. I. Hong, B. Gallagher, and T. Eliassi-Rad. Graphite: A visual query system for large graphs. In *ICDM Workshops*, pages 963–966, 2008.

[7] Y. J. Chu and T. H. Liu. On the shortest arborescence of a directed graph. *Science Sinica*, 14:1396–1400, 1965.

[8] C. Faloutsos, K. S. McCurley, and A. Tomkins. Fast discovery of connection subgraphs. In *KDD*, 2004.

[9] G. W. Flake, S. Lawrence, and C. L. Giles. Efficient identification of web communities. In *KDD*, 2000.

[10] M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *PNAS*, 99(12), 2002.

[11] P. Grünwald. *The Minimum Description Length Principle*. MIT Press, 2007.

[12] C. S. Helvig, G. Robins, and A. Zelikovsky. An improved approximation scheme for the group steiner problem. *Networks*, 37(1):8–20, 2001.

[13] W. Jin, R. K. Srihari, and X. Wu. Mining concept associations for knowledge discovery through concept chain queries. In *PAKDD*, pages 555–562, 2007.

[14] J. F. R. Jr., H. Tong, A. J. M. Traina, C. Faloutsos, and J. Leskovec. Gmine: A system for scalable, interactive graph visualization and mining. In *VLDB*, pages 1195–1198, 2006.

[15] G. Karypis and V. Kumar. Multilevel algorithms for multi-constraint graph partitioning. In *Supercomputing*, 1998.

[16] Y. Koren, S. C. North, and C. Volinsky. Measuring and extracting proximity in networks. In *KDD*, 2006.

[17] J. Leskovec, S. Dumais, and E. Horvitz. Web projections: learning from contextual subgraphs of the web. In *WWW*, pages 471–480, 2007.

[18] M. Li and P. Vitányi. *An Introduction to Kolmogorov Complexity and its Applications*. Springer, 1993.

[19] A. Liekens, J. De Knijf, W. Daelemans, B. Goethals, P. De Rijk, and J. Del Favero. BioGraph: unsupervised biomedical knowledge discovery via automated hypothesis generation. *Genome Biol.*, 12(6), 2011.

[20] V. Melkonian. New primal-dual algorithms for steiner tree problems. *Comp. Oper. Res.*, 34:2147–2167, 2007.

[21] C. Ramakrishnan, W. H. Milnor, M. Perry, and A. P. Sheth. Discovering informative connection subgraphs in multi-relational graphs. *SIGKDD Explor.*, 7(2):56–63, 2005.

[22] J. Riedy, D. A. Bader, K. Jiang, P. Pande, , and R. Sharma. Detecting communities from given seeds in social networks. *Technical Report GT-CSE-11-01*, 2011.

[23] J. Rissanen. Modeling by shortest data description. *Automatica*, 14(1):465–471, 1978.

[24] P. Sevon and L. Eronen. Subgraph queries by context-free grammars. *J. Integr. Bioinf.*, 5(2), 2008.

[25] D. Shahaf and C. Guestrin. Connecting the dots between news articles. In *KDD*, pages 623–632, 2010.

[26] W. Shrum, N. H. Cheek, and S. M. Hunter. Friendship in school: Gender and racial homophily. *Soc. Edu.*, 61:227–239, 1988.

[27] D. A. Spielman and S.-H. Teng. A local clustering algorithm for massive graphs and its application to nearly-linear time graph partitioning. *CoRR*, abs/0809.3232, 2008.

[28] H. Tong and C. Faloutsos. Center-piece subgraphs: problem definition and fast solutions. In *KDD*, pages 404–413, 2006.

[29] H. Tong, C. Faloutsos, and Y. Koren. Fast direction-aware proximity for graph mining. In *KDD*, pages 747–756, 2007.

[30] H. Tong, S. Papadimitriou, C. Faloutsos, P. S. Yu, and T. Eliassi-Rad. Basset: Scalable gateway finder in large graphs. In *PAKDD*, pages 449–463, 2010.

[31] W. Zachary. An information flow model for conflict and fission in small groups. *J. Anthr. Res.*, 33:452–473, 1977.

[32] A. Zelikovsky. A series of approximation algorithms for the acyclic directed steiner tree problem. *Algorithmica*, 18(1):99–110, 1997.