Efficient Algorithms for Mining Significant Substructures in Graphs with Quality Guarantees

Huahai He Ambuj K. Singh
Department of Computer Science
University of California, Santa Barbara
Santa Barbara, CA 93106, USA
{huahai, ambuj}@cs.ucsb.edu

Abstract

Graphs have become popular for modeling scientific data in recent years. As a result, techniques for mining graphs are extremely important for understanding inherent data and domain characteristics. One such exploratory mining paradigm is the k-MST (minimum spanning tree over k vertices) problem that can be used to discover significant local substructures. In this paper, we present an efficient approximation algorithm for the k-MST problem in large graphs. The algorithm has an $O(\sqrt{k})$ approximation ratio and $O(n \log n + m \log m \log k + nk^2 \log k)$ running time, where n and m are the number of vertices and edges respectively. Experimental results on synthetic graphs and protein interaction networks show that the algorithm is scalable to large graphs and useful for discovering biological pathways. The highlight of the algorithm is that it offers both analytical guarantees and empirical evidence of good running time and quality.

1 Introduction

Recent technological and scientific advances have resulted in an abundance of data that describe and model phenomena in terms of graphs. Graphs have been used for modeling hypertext, chemical compounds, biological pathways, protein structures and interactions, social networks, and taxonomies. For example, a metabolic pathway [19] is modeled as a set of reactions, enzymes, and metabolites, and an edge is placed between a reaction and a metabolite (or enzyme) if it participates in the reaction. Similarly, the 3D structure of proteins can be modeled as contact maps [17]: atoms whose distance is less than a threshold have an edge between them. Schema of heterogeneous web-based data sources and e-commerce sites or video data scenes [23] can also be modeled as graphs. A large proportion of the resulting graphs are annotated and weighted. The weights on edges can capture the physical proximity of the end nodes (contact maps, video scenes), frequency of their co-occurrence (hypertext), propensity of their interaction (pathways), or their similarity measured in other ways (social networks).

Given such an abundance of graphs derived from different application domains, there is a need for new techniques that can explore and mine graphs. Such techniques have the potential to advance our understanding in numerous ways: discovery of topological properties and connectivity patterns [1, 13, 28], understanding of evolutionary changes [5, 24], and isolation of network motifs [8, 25]. Many graph mining problems involve the discovery of substructures that have some significant properties. The substructures take the shape of paths [31], trees [9], clusters (cliques) [10, 26], or general graph patterns [8, 12, 18, 20, 25, 32, 33]. The metric for evaluating a substructure can be expressed in terms of weights on the paths or trees, density of the clusters, or frequency of the recurring graph patterns. Additional constraints can be added, such as size of the substructures, or specific vertices, edges, or subgraphs to be included. Most of these mining problems turn out to be intractable in the context of graphs. This has led to the development of heuristics that are efficient on large problem sizes and produce good results based on empirical evidence. While recognizing the need for scalability, we adopt a different approach toward measuring the quality: *We develop efficient heuristics that are good not just on account of empirical evidence but also because a provable bound exists on the quality of their results compared to the optimal solution (measured as an approximation ratio).*

The specific problem we consider in the context of the mining of local maximal structures is the k-*MST* problem: *Given a weighted graph, find a minimum weight tree spanning k vertices.* The k-*MST* problem can be used for exploring a well-connected set of nodes in weighted graphs. For example, it can be used for finding pathways in stochastic biological networks, a group of people in a social network, or a set of amino acids in physical proximity in a protein structure. There are a number of variants of the basic k-*MST* problem. One might specify a vertex of interest to be included in the k-MST, leading to the *rooted* version of the problem. Additionally, one may specify other constraints such as the type of nodes or edges to include in the tree.

The k-*MST* problem has been of historical interest. Since it has been proved NP-hard [29], the focus though has been on improving the approximation ratio with little regard to the running time (although polynomial). Thus, the approximation ratio has been improved from $O(\sqrt{k})$ to $O(log^2k)$ [4], $O(logk)$ [27], 17 [7], 3 [14], 2.5 [3], $2+\epsilon$ [2], and recently to 2 [15]. However, all the existing approximation algorithms have a non-trivial time complexity, greater than $O(n^2)$, where n is the number of nodes. For very large graphs, this time complexity is prohibitive, especially for exploratory interactive mining that requires an instant response. More importantly, these algorithms are designed just for ensuring the approximation factor. They rely on complicated routines such as primal-dual methods.

We present an efficient approximation algorithm for rooted k-MST in large graphs. The algorithm has an $O(\sqrt{k})$ approximation ratio and an $O(n \log n + m \log m \log k + nk^2 \log k)$ time complexity for the rooted version of the problem, where m is the number of edges (we assume k is much smaller than n). The algorithm easily outperforms existing algorithms in terms of time complexity and is the first known implementation for the k-MST problem. Moreover, in the process of developing our algorithm, we define two new problems, that of k-MST in arbitrary unrooted, undirected trees and subset selection in hierarchical clusters. Both problems are solved optimally in tree structures using dynamic programming (DP). Specifically, we make the following contributions:

1. We present a double dynamic programming technique for k-MST in an arbitrary unrooted, undirected tree (a.k.a. acyclic undirected graph). This technique finds the optimal k-MST in $O(mk^2)$ running time. The technique differs from existing approaches which have to transform the tree into a rooted binary tree [11, 21] or decomposable graphs [29].

2. We define the problem of subset selection in hierarchical clusters: choose a set of disjoint clusters that have exactly (or at least) k vertices. A dynamic programming based technique is presented to find the optimal subset of clusters. This technique is the main building block of our approximation algorithm for k-MST in graphs, yet it is a general problem of interest on its own.

3. We present an approximation algorithm for rooted k-MST in graphs with $O(n \log n + m \log m \log k + nk^2 \log k)$ time complexity and $O(\sqrt{k})$ approximation ratio.

Experimental results on synthetic graphs and protein interaction networks validate the scalability and quality of the approximation algorithm. The algorithm takes only seconds for graphs having about 100K vertices and 500K edges. The empirical quality is consistent with the theoretical $O(\sqrt{k})$ approximation ratio. The algorithm outperforms an alternative approach based on shortest paths significantly in terms of quality. We also apply the algorithm to discover biological pathways in a yeast network. In particular, we found a pathway which is consistent with the reference MAPK signaling pathways. This validates the practical usefulness of the algorithm.

The rest of this paper is organized as follows. Section 2 presents double dynamic programming for k-MST in trees. Section 3 presents our approximation algorithm for k-MST in graphs. Section 4 proves the approximation guarantee of our algorithm. Experimental results are reported in Section 5. Section 6 discusses related work and Section 7 concludes the paper.

2 K-MST in Trees

When the input graph is a tree, the k-MST problem can be solved in polynomial time. In this section, we present a double dynamic programming technique for k-MST in trees. The technique will be used in our approximation algorithm for k-MST in graphs.

Table 1. Symbols used in the paper

Symbol	Description
n	Number of nodes
m	Number of edges
k	Number of vertices in the k-MST
u_0	Root vertex of the k-MST
OPT	Cost of the optimal k-MST
$c(u)$	Cost of node u
$c(u,v)$	Cost of edge (u,v)
$F(u,v,l)$	Cost of the optimal subtree that contains u but not v, of size l
d	Degree of node u minus one
$r_1..r_d$	Neighbor nodes of u, exclusive of v
$A(t,y)$	Cost of the optimal forest from the first t branches $r_1..r_t$ having y nodes

2.1 Outer Dynamic Programming

Figure 1 illustrates the idea of outer dynamic programming. Let u be a node and v be a neighbor node of u, $F(u, v, l)$ be the cost of the optimal subtree of size l that contains u but not v. The size of a tree is defined as the number of nodes in the tree. Let $r_1, ..., r_d$ be the neighbor nodes of u exclusive of v. Then, the optimal subtree consists of node u and d branches where each branch is either empty or is another optimal subtree of size l_i that contains r_i but not u. Thus, a recurrence relation can be established as

$$F(u, v, l+1) = c(u) + \min_{l_i \geq 0, \sum_1^d l_i = l} \sum_{i=1}^{d} F'(r_i, u, l_i), l \geq 0$$
$$F(u, v, 1) = c(u) \qquad (1)$$

where $c(u)$ is the cost of node u and

$$F'(r, u, l) = \begin{cases} 0, & \text{if} \quad l = 0 \\ F(r, u, l) + c(r, u), & \text{if} \quad l > 0 \end{cases}$$

is the cost of the optimal subtree associated with (r, u, l) plus the cost of edge (r, u). In other words, we can assemble a larger optimal subtree by combining smaller optimal subtrees.

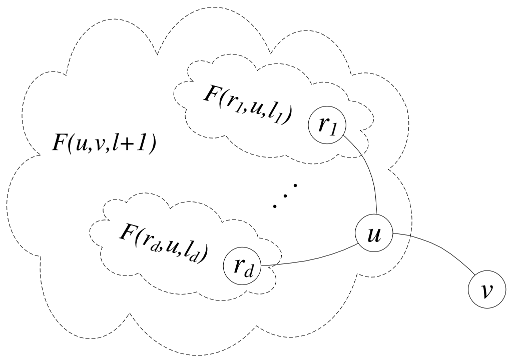

Figure 1. Illustration of outer dynamic programming

In Eqn. (1), the number of combinations for splitting l into d parts is $\binom{l+d-1}{d-1}$. For rooted binary trees, this number is small. For arbitrary trees where the degree of a node is unconstrained, a naive approach is not efficient. Fortunately, we can find the optimal combination using dynamic programming again, namely *Inner Dynamic Programming*.

2.2 Inner Dynamic Programming

Figure 2 illustrates the idea of inner dynamic programming. Let $A(t, y)$ be the optimal forest from the first t branches $r_1..r_t$ having y nodes. Then $A(t, y)$ can be computed by considering the optimal forest from the first $t-1$ branches and the optimal subtree from the t^{th} branch:

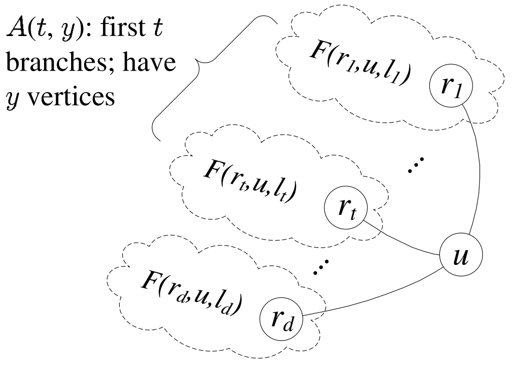

Figure 2. Illustration of inner dynamic programming

$$A(t, y) = \min_{0 \leq x \leq y} \{A(t-1, x) + F'(r_t, u, y-x)\}, t > 1$$
$$A(t, 0) = 0 \qquad (2)$$
$$A(1, y) = F'(r_1, u, y)$$

Using the inner dynamic programming, Eqn. (1) can then be computed by

$$F(u, v, l+1) = c(u) + A(d, l) \qquad (3)$$

For the rooted version of k-MST, i.e., the k-MST must contain a specific node u_0, all neighbors of u_0 are taken into account at the final iteration of the outer DP:

$$F(u_0, k) = c(u_0) + \min_{l_i \geq 0, \sum_1^{d+1} l_i = k-1} \sum_{1}^{d+1} F'(r_i, u_0, l_i) \qquad (4)$$

or

$$F(u_0, k) = c(u_0) + \min_{0 \leq x \leq k-1} \{A(d, x) + F'(v, u_0, k-1-x)\} \qquad (5)$$

The time complexity is analyzed as follows. The outer DP computes k entries for each edge, totally $O(mk)$ entries. For each entry $F(u, v, l+1)$, the inner DP updates matrix $A(\cdot, \cdot)$ by computing a new column $A(i, l)$ for $i = 1..d$. This update takes $O(dk)$ time. Note that the degrees $d's$ sum up to $2m$. Therefore, the total time complexity is $O(mk^2)$. The space complexity is also $O(mk^2)$.

3 Approximate K-MST in Graphs

In this section, we present the approximation algorithm for k-MST in graphs. Our algorithm actually solves the rooted version of k-MST. We first describe subset selection on hierarchical clusters, the main building block of our algorithm, and then present the approximation algorithm and its time complexity.

3.1 Subset Selection on Hierarchical Clusters

Given a graph and a hierarchical clustering of its vertices, we can select a subset of clusters that have exactly (or at least) k vertices. If we assign a cost to each cluster, can we find the subset of clusters that minimizes the total cost? Since the hierarchical clustering forms a rooted tree structure, this problem can be solved optimally using dynamic programming as well.

GOAL. Given a rooted binary tree, each tree node (corresponding to a cluster) has a given cost. The cost of a subset of nodes is the sum of costs of the nodes. Nodes are disjoint if they do not form an ancestor-descendent relationship. The goal is to find the minimum cost subset of disjoint nodes subject to the constraint that these nodes have exactly (or at least) k leaves in total. An example of the subset selection problem is shown in Figures 3 and 4.

DYNAMIC PROGRAMMING. Let C be a tree node, C_L and C_R be the left and right child node of C, $F(C, k)$ be the cost of the optimal subset under C having exactly k leaves. Then,

$$F(C,k) = \min_{0 \le i \le k} \{F(C_L, i) + F(C_R, k-i)\} \qquad (6)$$

If the number of leaves under C is equal to k, then the set consisting of node C only is considered as well:

$$F(C,k) \leftarrow \min\{F(C,k),\ Cost(C)\}, \quad \text{if} \quad \#\text{leaves}(C) = k \qquad (7)$$

Once a node is selected, none of its descendent nodes are selected. Thus, the subset of nodes is always disjoint.

For leaves, the costs are trivially defined by

$$F(\text{leaf}, 0) = 0$$
$$F(\text{leaf}, 1) = Cost(\text{leaf})$$
$$F(\text{leaf}, i) = +\infty, \quad \forall\, i > 1$$

To find the optimal subset of nodes having *at least* k leaves, let $H(C, k)$ be the cost of the optimal subset of nodes having at least k leaves. Then,

$$H(C,k) = \min_{0 \le i \le k} \{F(C_L, i) + H(C_R, k-i),\ H(C_L, i) + F(C_R, k-i)\} \qquad (8)$$

If node C has at least k leaves, then

$$H(C,k) \leftarrow \min\{H(C,k),\ Cost(C)\}, \quad \text{if} \quad \#\text{leaves}(C) \ge k \qquad (9)$$

The dynamic programming is carried out from bottom to top. The total time complexity is $O(nk)$ where n is the number of tree nodes. For arbitrary rooted trees, one can use an inner dynamic programming in a similar way as in Section 2.

Although designed for our approximation algorithm for k-MST, the problem of subset selection on hierarchical clusters is general in its own and can be used for other scenarios. Variants of the problem include constraining the number of clusters instead of the number of vertices, or constraining both of them. These variants can also be solved by dynamic programming.

3.2 Approximation Algorithm for k-MST

Our approximation algorithm first performs a hierarchical clustering, selects the best subset of clusters having k vertices, connects them to the root vertex, and finally extracts the optimal k-MST from the resultant tree.

To guarantee the theoretical approximation ratio, we need to guess the optimal solution OPT. Let L_{low} be the distance from root vertex u_0 to the k^{th} nearest vertex, L_{up} be the sum of edges in the shortest paths that connect the first k vertices. Clearly, L_{low} and L_{up} are the lower and upper bound of OPT respectively, and $L_{up} \le kL_{low}$.

The approximation algorithm for k-MST is as follows.

1. Compute the shortest distance from each vertex to the root vertex u_0 using Dijkstra's algorithm.

2. Guess the bound of the optimal cost OPT. For $L = L_{low}, 2L_{low}, 4L_{low}, ..., L_{up}$, consider only vertices whose distance is within L, repeat steps 3-7 and report the best k-MST found.

3. Single linkage hierarchical clustering (Kruskal's algorithm): initially every vertex is a cluster. At each iteration, pick up the next minimum edge that connects two clusters, which forms a new super cluster. Repeat until all clusters are connected (which forms a single large cluster). Each cluster is represented by the minimum tree spanning its vertices.

4. For each cluster C_i, compute its distance from the root vertex. This can be recursively computed from bottom to top. Define the cost of cluster C_i as the edge weights of its spanning tree plus its distance to the root vertex, $Cost(C_i) = \pi(C_i) + d(C_i, u_0)$.

5. Denote the set of all clusters by $\mathcal{S}$. Find the optimal subset of disjoint clusters $\mathcal{C}$ that have exactly k ver-

tices, i.e.,

$$\mathcal{C} = \arg\min_{\mathcal{C}\subseteq\mathcal{S}} \{ \sum_{C_i \in \mathcal{C}} Cost(C_i) \}$$

$$\text{subject to} \quad \sum_{C_i \in \mathcal{C}} |C_i| = k$$

This step is done by subset selection on hierarchical clusters described in Section 3.1.

6. For each cluster C_i in $\mathcal{C}$, connect C_i to the root vertex using the shortest path. The shortest path may connect to other clusters or shortest paths before it reaches the root vertex. In that case, we only need to connect the cluster to whichever cluster (or vertex in other shortest paths) is closest to it. The result of this step is a tree having at least k vertices.

7. Find the optimal k-MST from the resultant tree using the technique described in Section 2.

Some aspects to note about the above algorithm:

1. Step 2 (guessing the bound of OPT) is used to guarantee the approximation ratio.

2. In step 5 (subset selection of clusters), the goal could be changed to finding the optimal subset of clusters having *at least* k vertices.

3. Step 7 extracts the best k-MST from the resultant tree. If one extracts a simple k-subtree, the approximation ratio is still preserved. And the time complexity will be reduced from $O(nk^2)$ to $O(k)$. This could be useful for large k.

The time complexity is analyzed as follows. Step 1 takes $O((m+n)\log n)$ (or $O(m+n\log n)$ if using the Fibonacci heap). Step 2 repeats $\log k$ times. Step 3 takes $O(m \log m)$ time. Step 4 takes $O(n)$ time. Step 5 takes $O(nk)$ time. Step 6 takes $O(n)$ time. Step 7 takes $O(nk^2)$ time (or $O(k)$ if extracting a simple k-subtree). Thus, the total time complexity is $O(n \log n + m \log m \log k + nk^2 \log k)$. Since $k \ll n$ in practice, the algorithm is scalable to large graphs. This is validated in the experiments.

In addition to the time efficiency, the algorithm is highly adaptable and allows for variants: 1) Instead of single linkage clustering, other hierarchical clustering can be used. 2) Other cost functions can be used for subset selection of clusters. 3) Vertices can have weights. 4) Multiple roots can be allowed (i.e., Steiner trees).

We illustrate the algorithm using an example. Figure 3 shows an edge weighted graph. Figure 4 shows a hierarchical clustering of the graph. Leaves are shown in ellipses and internal nodes are shown in rectangles. Now, suppose one wants to find a k-MST rooted at v_0 with size of 4. By Steps 3 and 4 of the algorithm, the cost of each cluster is computed as the the weight of the cluster plus its distance to v_0. The costs are shown in the tree nodes of Figure 4. Next, Step 5 selects the optimal subset of clusters having 3 vertices. The selected subset of clusters is $\{C_1, v_4\}$. It can be verified that this subset outperforms other candidates, such as $\{C_4\}$ and $\{C_2, v_1\}$. The selected clusters are then connected to the root vertex. Figure 5 shows the resultant tree. As this tree already has exactly 4 vertices, the algorithm terminates.

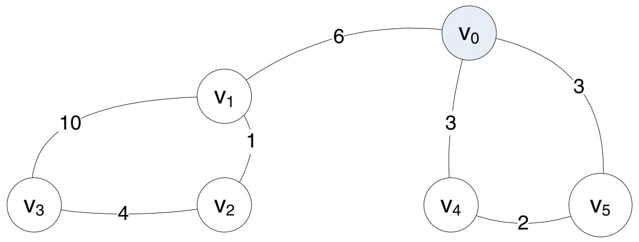

Figure 3. An example graph

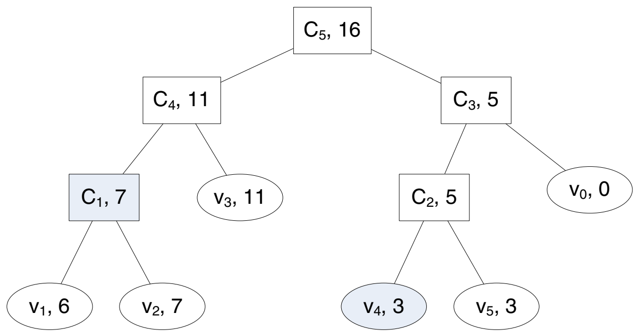

Figure 4. Subset selection on hierarchical clusters with $k = 3$. The optimal subset of clusters is $\{C_1, v_4\}$ with a total cost of 10. The corresponding vertices are $\{v_1, v_2, v_4\}$

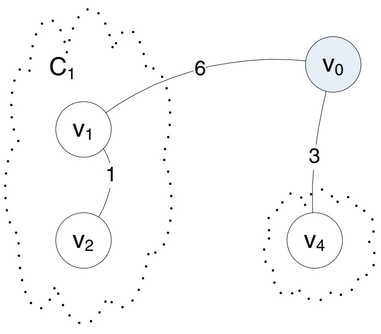

Figure 5. The k-MST resulting from Figure 4

4 Approximation Ratio

In this section we prove the quality of our approximation algorithm. Let OPT be the cost of the optimal k-MST. We have the following approximation guarantee:

Theorem 1. *The cost of the approximate k-MST is at most $O(\sqrt{k}) \cdot OPT$.*

To prove Theorem 1, we first show that there always exists a small subset of clusters that has k leaves.

Lemma 1. *Given a rooted binary tree having n leaves, for any k and l, $n \geq k \geq l \geq \log k$, there exist l nodes (either internal nodes or leaves) such that the total number of leaves under these l nodes is equal to k.*

Proof of Lemma 1. (By construction) We allocate k recursively from top to bottom. At each node, if one subtree has at least k leaves, then allocate k into that subtree. Otherwise, allocate fully to the larger subtree, and the rest of k into the smaller subtree. Let L and R be the number of leaves under the larger and smaller subtree. We have $k \leq L + R \leq 2L$. Then, $k - L \leq k/2$, i.e., the rest allocated into the smaller subtree must be at most $k/2$. In the end, there are at most $\log k$ fully allocated subtrees. These subtrees cover k leaves.

Next, we collapse nodes bottom-up under these fully allocated subtrees. A node can be collapsed if both of its child nodes are leaves. The node then become a new "leaf". Repeating this process will cause the number of "leaves" to decrease gradually. Eventually there will be $\log k$ "leaves" left. Since $k \geq l \geq \log k$, at some stage, we can find l "leaves" left. These l "leaves" correspond to l nodes in the original tree, which cover k leaves. $\square$

The lemma can be generalized to rooted trees with maximum fan-out of Δ, and l is limited by $k \geq l \geq (\Delta - 1)\log k$.

Now, we prove the approximation ratio in Theorem 1. The basic idea is to decompose the optimal k-MST into a set of clusters, and then infer the existence of a feasible solution (a subset of clusters in the clustering hierarchy) such that: each intra-cluster edge is bounded by $O(\frac{OPT}{\sqrt{k}})$ and as a result, their sum is bounded by $O(\sqrt{k} \cdot OPT)$; each inter-cluster edge is bounded by $O(OPT)$ but the number of clusters is at most $O(\sqrt{k})$.

Proof of Theorem 1. Let w be the threshold edge weight below which the optimal k-MST is divided into $\sqrt{k}$ clusters, $C_1^*..C_{\sqrt{k}}^*$. Then, the number of inter-cluster edges is $\sqrt{k}$ and $w \leq \frac{OPT}{\sqrt{k}}$.

Now, consider the optimal clusters $C_1^*..C_{\sqrt{k}}^*$ and clusters in the clustering hierarchy. Each C_i^* is not necessarily a cluster in the clustering hierarchy. However, its vertices must be contained in some super-cluster C_i in the clustering hierarchy below the same w threshold. The following properties can be observed: 1) the number of such super-clusters $C_i's$ is at most $\sqrt{k}$; 2) the total number of vertices under these super-clusters is at least k; 3) The weight of each edge in the super-clusters is no more than w. Figure 6 illustrates the optimal clusters and the super-clusters.

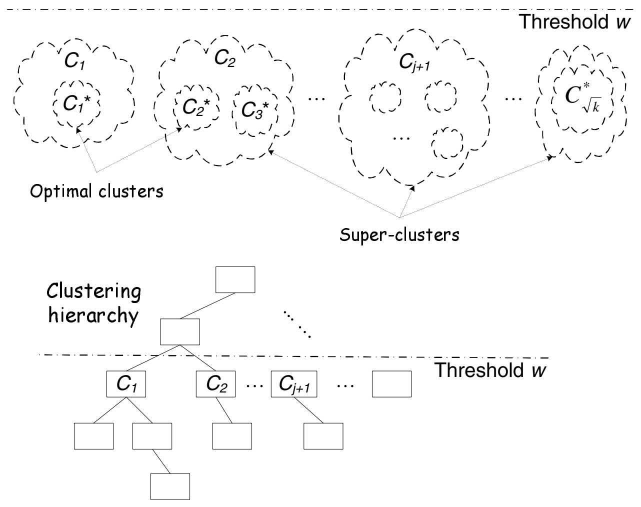

Figure 6. Illustration of the clusters

Next, we show that there exist $O(\sqrt{k})$ clusters in the clustering hierarchy which cover exactly k vertices. Let S_i be the total number of leaves in the first i super-clusters $C_1, ..., C_i$. Then, there exists $j < \sqrt{k}$ such that $S_j < k \leq S_{j+1}$. We select the first j super-clusters $C_1, ..., C_j$, and then select sub-clusters from super-cluster C_{j+1} such that they cover $k - S_j$ vertices. According to Lemma 1, the number of these sub-clusters is bounded by $\log(k - S_j)$. Thus, there exist $O(\sqrt{k})$ clusters in the clustering hierarchy which cover exactly k vertices.

As a result, these $O(\sqrt{k})$ clusters form a feasible solution in the search space of the algorithm (Step 5: subset selection on hierarchical clusters). The sum of intra-cluster edges of this solution is bounded by $\sqrt{k} \cdot OPT$. For inter-cluster edges, the guessing step (step 2) guarantees two trials L and $2L$ where $L \leq OPT \leq 2L$. The trial $2L$ ensures that all distances from the root vertex to the clusters (and in turn inter-cluster edges) are bounded by $2\,OPT$. Thus, the total cost of this feasible solution is bounded by $O(\sqrt{k}) \cdot OPT$.

Since the algorithm always finds the optimal subset of clusters, the cost of the optimal subset is also bounded by the above feasible solution. Therefore, the algorithm guarantees an $O(\sqrt{k})$ approximation ratio. $\square$

5 Experimental Results

In this section, we evaluate our approximation algorithm on both synthetic and real graphs, and apply the algorithm to pathway discovery of biological networks. Section 5.1 evaluates the performance and quality of the algorithm on synthetic graphs. Section 5.2 reports results on protein interaction networks and compares the discovered pathways to reference pathways.

We measure the quality of results by comparison to OPT. Since calculating OPT is intractable, we compute a naive lower bound of OPT which is the distance from the k^{th} nearest vertex to the root (L_{low} from Section 3.2). The *upper bound of approximation ratio* is defined by the ratio of the algorithm's cost to L_{low}.

Since no implementations of the state-of-the-art algorithms exist, we empirically compare our algorithm with a shortest path approach: find the first k nearest vertices, and then connect them to the root to form a subtree of size k (L_{up} from Section 3.2). The shortest path approach has an $O(k)$ approximation ratio.

All the algorithms were implemented in Java using Sun JDK 1.6. All experiments were performed on an Intel 2.8GHz, 2G memory running MS Win XP.

5.1 Synthetic Graphs

The synthetic graphs are generated using a simple random graph model: given n and m, first generate a random tree that connects n vertices, and then generate the rest of edges by randomly choosing two end points. (More realistic models [1, 13, 24] could also be used for generating the synthetic graphs.) To generate the edge weight, we first generate a random probability value, and then convert it to an edge weight by taking the negative log.

We first fix n and m at 20K and 100K respectively, and vary k as [10, 30, 50, 70, 90]. For each k, we randomly select a vertex as the root and run the approximation algorithm. 1000 trials are conducted and the average results are reported.

Figure 7 shows the running time with varying k. The total running time is broken down to four phases: 1) Shortest path computation, 2) Hierarchical clustering, 3) Cluster selection, and 4) Finding k-MST in the resultant tree. Phase 1 is independent of k and depends only on the graph. Phase 2 is repeated at most $\log k$ times. Phase 3 takes time linear in k, which is consistent with the time complexity of cluster selection. This step dominates the running time for high values of k. Phase 4 has a time complexity of $O(nk^2)$. In practice, the size of the resultant tree is much smaller than n. Thus, the running time of this phase is smaller than the other phases. Overall, the running time is in seconds, indicating the efficiency of the approximation algorithm.

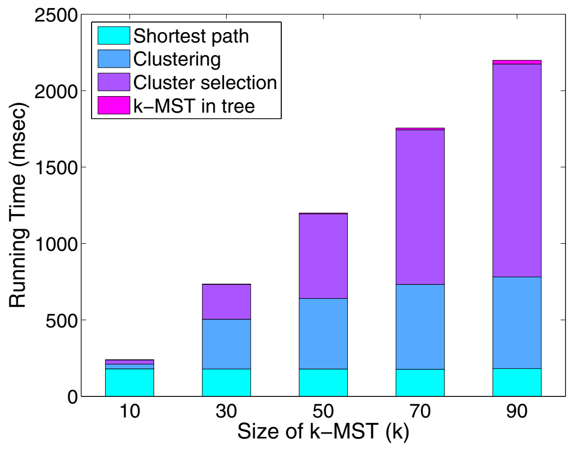

Figure 7. Running time with various k for synthetic graphs

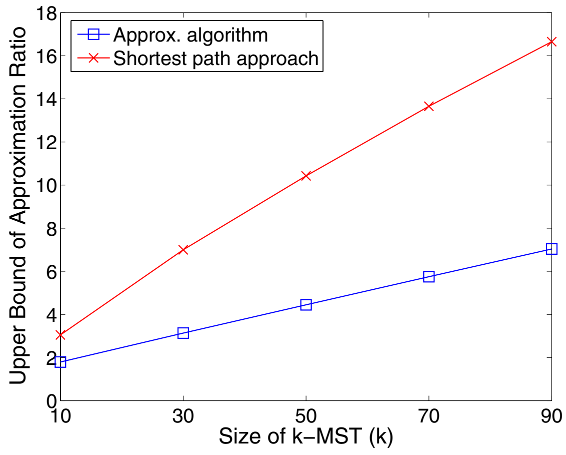

Figure 8. Upper bound of approximation ratio with various k for synthetic graphs

Figure 8 shows the upper bound of approximation ratio with varying k. As can be seen in the figure, the approximation ratio of our algorithm is much smaller than that of the shortest path approach. The ratio grows slowly with increasing k. This validates the quality of our algorithm.

Next, We vary the number of vertices n as $[20K, 40K, 60K, 80K, 100K]$ and $m = 5 * n$, and fix k at 10 and 100 respectively. Figure 9 shows the running time with varying n. As the graphs become larger, the running time increases accordingly. However, the running time is still in seconds when the graph contains 100K vertices. The results validates the scalability of our algorithm.

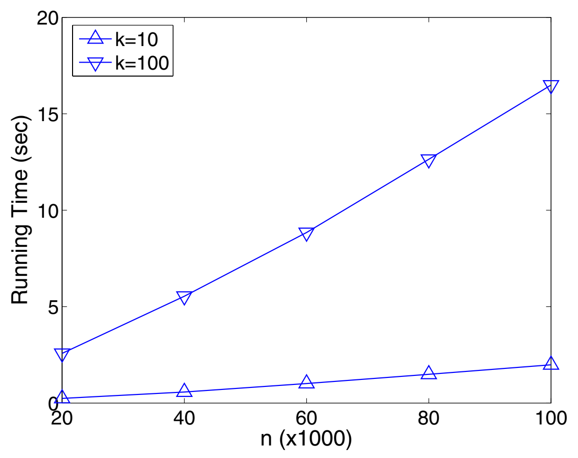

Figure 9. Running time with various sizes of synthetic graphs

5.2 Detection of Biological Pathways

We use the yeast (*S. cerevisiae*) probabilistic network by Lee et al. [22] which contains 5552 vertices and 34000 edges. The probabilities on edges are converted into edge weights by taking the negative log.

We apply our algorithm for finding biological pathways in the yeast network. We consider the known MAPK signaling pathways in yeast as the reference pathways. Figure 10 (a) shows the pheromone and filamentous growth pathways adapted from [30] and the KEGG database. Note that the two pathways share the MAPK chain "CDC42-STE20-STE11-STE7" and the transcription factor "STE12".

We choose the receptor "STE2" in the pheromone pathway as the root vertex. By setting k at 10, we found a pathway from the yeast network as shown in Figure 10 (b). All vertices in the discovered pathway except "SST2" show up in the reference pathways. Interestingly, the discovered pathway contains "KSS1", which is present in the filamentous growth pathway. "SST2" does not exist in the reference pathways. This protein is responsible for activating "GPA1" (http://www.yeastgenome.org). It regulates desensitization to alpha factor pheromone. It is also required to prevent receptor-independent signaling of the mating pathway.

The tree structure of the discovered pathway does not completely match the reference pathways. This is due to the limitation of the input graph. For instance, there is no edge between "STE18" and "CDC42" in Lee's network. However, the algorithm is still able to circumvent the broken chain to find the rest of the vertices.

Figure 11 shows the induced subgraph of Figure 10 (b) in Lee's network. The induced subgraph forms a clique-like pattern. By comparison, the tree-like pattern has a few advantages over the clique-like pattern. First, it is easier

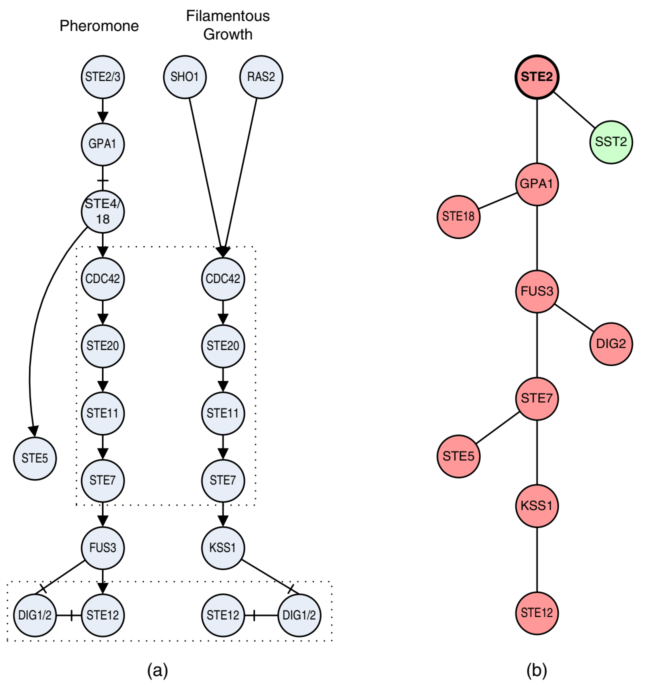

Figure 10. MAPK signaling pathways in yeast. (a) The known pheromone and filamentation pathways, adapted from [30] and KEGG (http://www.genome.jp/kegg). (b) the pathway discovered from Lee's network by our algorithm with root="STE2" and $k = 10$.

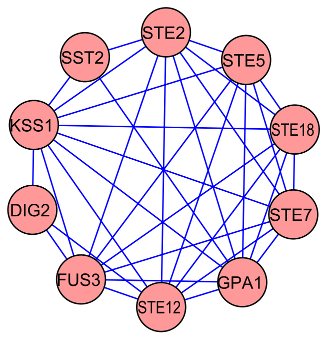

Figure 11. The induced subgraph of the discovered pathway (Figure 10 (b)) in Lee's network

to examine and explain the biological meaning of a tree-like pattern than that of a clique-like pattern. The tree-like pattern has a smaller number of edges and defines how the proteins interact in the pathway. Second, a tree-like pattern may span vertices in multiple clusters. In contrast, techniques for clique discovery tend to exclude vertices not in the same cluster. Inter-cluster interactions are ignored. Finally, when the network itself does not include cliques (e.g., signal transduction pathways), tree-based pattern finding algorithms will perform better.

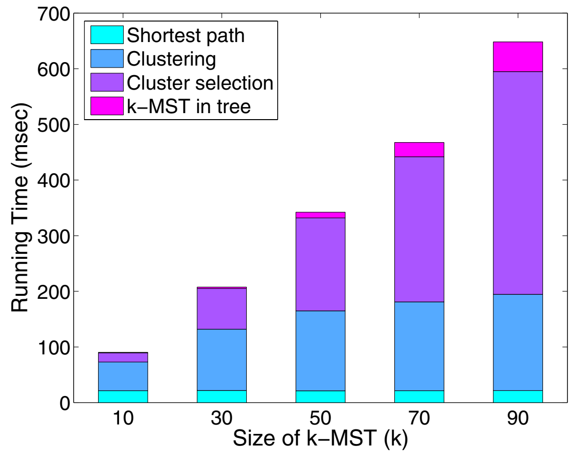

Figure 12. Running time with various k for the yeast network

We also evaluate the running time of our algorithm on the yeast network. Figure 12 shows the running time with varying k. The breakdown across the different phases is similar to the case of synthetic graphs. The total running time is in seconds. This validates the practical efficiency of our algorithm on real graphs.

6 Related Work

APPROXIMATION ALGORITHMS FOR k-MST. The k-MST problem has been studied in the algorithm community for a long time. Ravi et al. [29] gave the first non-trivial algorithm for unrooted k-MST with $O(\sqrt{k})$ approximation ratio and $O(n^2(m + n \log n))$ running time. The algorithm consists of two phases. The merge phase generates clusters by single linkage hierarchical clustering. The merge phase continues until there exists a cluster having between k and $2k$ vertices. A new instance of collect phase is entered whenever there exist at most $\sqrt{k}$ clusters having at least k vertices. The approximation guarantee lies in that the weight of each intra-cluster edge is bounded by $OPT/\sqrt{k}$ ($\sqrt{k} \cdot OPT$ in sum) and the number of inter-cluster edges is bounded by $\sqrt{k}$ ($\sqrt{k} \cdot OPT$ in sum), where OPT is the cost of the optimal k-MST. Our proof of Theorem 1 is inspired by this approach.

Awerbuch et al. [4] improved the approximation ratio to $O(\log^2 k)$ for the rooted version. They merged clusters using a new measure $\frac{d(C_i,C_j)}{min(|C_i|,|C_j|)}$, i.e., large and nearby clusters are preferred for merging. The merge phase is invoked several times and each time a cluster is found and connected to the root. At most $O(\log k)$ clusters are required such that the clusters have at least k vertices. The algorithm needs to compute the distance between every pair of clusters. The time complexity is at least $O(n^2 \log n)$.

Blum et al. [7] gave the first constant factor 17-approximation algorithm for rooted k-MST with an $O(n^2 \log^4 n)$ running time. Garg [14] improved the factor to 3 for rooted k-MST. Based on Garg's algorithm, Arya and Ramesh [3] presented a 2.5 approximation algorithm. Arora and Karakostas [2] presented a 2+ϵ approximation algorithm. Recently, Garg [15] improved the approximation ratio to 2 for unrooted k-MST. All the constant factor approximation algorithms rely on the Goemans-Williamson algorithm [16] for the prize-collecting Steiner tree problem, which has a time complexity of $O(n^2 \log n)$.

DYNAMIC PROGRAMMING IN TREES. When restricted to trees, the k-MST problem can be solved in polynomial time using dynamic programming [29]. Fagin et al. [11] used dynamic programming to implement analytical operations on multi-structural databases. Kumar et al. [21] used dynamic programming for hierarchical topic segmentation of websites. All the techniques transform the tree into a rooted binary tree (or binary composition rules) before applying dynamic programming. In contrast, our double dynamic programming technique (Section 2) can be directly applied to arbitrary unrooted, undirected trees.

DETECTION OF BIOLOGICAL PATHWAYS. In the analysis of protein interaction networks, Scott et al. [31] proposed a randomized algorithm for detecting signaling pathways. They used a technique called color coding to find optimal simple paths of length k with probability e^{-k}. The problem is NP-hard since the traveling salesman problem can be reduced to it. Instead of approximating the optimal path, they repeat many trials until the algorithm has a high probability of finding the optimal path. The probability depends on the number of trials which is exponential in k.

CLUSTER DISCOVERY. A number of algorithms have been proposed recently for finding clusters or clique-like substructures in graphs [6, 10]. These algorithms are useful in finding substructures where each node is connected closely to all other nodes in the substructure. The k-MST problem on the other hand finds substructures that are branching or "tree-like". The difference between the two kinds of substructures can be illustrated by noting that the former corresponds to protein complexes in biological networks, while the latter corresponds to signaling, transcriptional, or metabolic pathways.

7 Conclusion

Graph mining has become an extremely important problem due to the exponential growth in scientific data of various kinds. We have presented an efficient algorithm for significant substructure discovery, specifically k-MST, in large graphs. The algorithm has an interesting framework for mining significant substructures: a hierarchical clustering step collects significant components; a selection step extracts an optimal subset of the components; finally, a significant substructure is assembled using the subset. Theoretical analyses establish a good time complexity and approximation ratio of the algorithm. Experimental results provide empirical evidence that the algorithm is scalable to large graphs and achieves a good quality. We have demonstrated the practical usefulness of the algorithm by finding pathways in biological interaction networks. Extensions to other substructure discovery problems while ensuring worst-case bounds on both running time and quality are planned for the future.

Acknowledgements: This work was supported in part by NSF grants IIS-0612327, DBI-0213903 and by the ARO ICB grant DAAD19-03-D-0004.

References

[1] R. Albert, H. Jeong, and A.-L. Barabasi. Diameter of the world-wide web. In *Nature*, pages 401:130–131, 1999.

[2] S. Arora and G. Karakostas. A $2 + \epsilon$ approximation algorithm for the k-MST problem. In *Proc. of ACM-SIAM symposium on Discrete algorithms (SODA '00)*, pages 754–759, 2000.

[3] S. Arya and H. Ramesh. A 2.5-factor approximation algorithm for the k-MST problem. *Information Processing Letters*, 65(3):117–118, 1998.

[4] B. Awerbuch, Y. Azar, A. Blum, and S. Vempala. Improved approximation guarantees for minimum-weight k-trees and prize-collecting salesmen. In *Proc. of ACM Symposium on Theory Of Computing (STOC)*, pages 277–283, 1995.

[5] L. Backstrom, D. P. Huttenlocher, J. M. Kleinberg, and X. Lan. Group formation in large social networks: membership, growth, and evolution. In *KDD*, pages 44–54, 2006.

[6] G. Bader and C. Hogue. An automated method for finding molecular complexes in large protein interaction networks. *BMC Bioinformatics*, 4:2, 2003.

[7] A. Blum, R. Ravi, and S. Vempala. A constant-factor approximation algorithm for the k-MST problem. In *Proc. of ACM symposium on Theory of computing (STOC '96)*, pages 442–448, 1996.

[8] J. Chen, W. Hsu, M.-L. Lee, and S.-K. Ng. NeMoFinder: dissecting genome-wide protein-protein interactions with meso-scale network motifs. In *KDD*, pages 106–115, 2006.

[9] Y. Chi, Y. Yang, and R. R. Muntz. Indexing and mining free trees. In *ICDM*, 2003.

[10] S. Dongen. *Graph Clustering by Flow Simulation*. Phd thesis, University of Utrecht, May 2000.

[11] R. Fagin, R. V. Guha, R. Kumar, J. Novak, D. Sivakumar, and A. Tomkins. Multi-structural databases. In *Proc. of PODS*, pages 184–195, 2005.

[12] C. Faloutsos, K. S. McCurley, and A. Tomkins. Fast discovery of connection subgraphs. In *Proc. of KDD '04*, 2004.

[13] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. In *SIGCOMM*, 1999.

[14] N. Garg. A 3-approximation for the minimum tree spanning k vertices. In *Proc. of Symposium on Foundations of Computer Science (FOCS '96)*, 1996.

[15] N. Garg. Saving an epsilon: a 2-approximation for the k-MST problem in graphs. In *Proc. of ACM symposium on Theory of computing (STOC '05)*, pages 396–402, 2005.

[16] M. X. Goemans and D. P. Williamson. A general approximation technique for constrained forest problems. *SIAM J. Comput.*, 24(2):296–317, 1995.

[17] J. Hu, X. Shen, Y. Shao, C. Bystroff, and M. J. Zaki. Mining protein contact maps. In *BIOKDD*, 2002.

[18] R. Jin, C. Wang, D. Polshakov, S. Parthasarathy, and G. Agrawal. Discovering frequent topological structures from graph datasets. In *KDD*, pages 606–611, 2005.

[19] KEGG. http://www.genome.ad.jp/kegg/.

[20] Y. Koren, S. C. North, and C. Volinsky. Measuring and extracting proximity in networks. In *Proc. of KDD '06*, 2006.

[21] R. Kumar, K. Punera, and A. Tomkins. Hierarchical topic segmentation of websites. In *Proc. of KDD '06*, pages 257–266, 2006.

[22] I. Lee, S. V. Date, A. T. Adai, and E. M. Marcotte. A probabilistic functional network of yeast genes. *Science*, 306:1555–1558, 2004.

[23] J. Lee, J. Oh, and S. Hwang. STRG-index: Spatio-temporal region graph indexing for large video databases. In *Proc. of SIGMOD*, 2005.

[24] J. Leskovec, J. Kleinberg, and C. Faloutsos. Graphs over time: Densification laws, shrinking diameters and possible explanations. In *KDD*, 2005.

[25] R. Milo et al. Network motifs: Simple building blocks of complex networks. *Science*, October 2002.

[26] J. Pei, D. Jiang, and A. Zhang. On mining cross-graph quasi-cliques. In *KDD*, pages 228–238, 2005.

[27] S. Rajagopalan and V. Vazirani. Logarithmic approximation of minimum weight k trees. In *Unpublished Manuscript*, 1995.

[28] E. Ravasz, A. L. Somera, D. A. Mongru, Z. N. Oltvai, and A.-L. Barabasi. Hierarchical organization of modularity in metabolic networks. *Science*, volume 297, August 2002.

[29] R. Ravi, R. Sundaram, M. Marathe, D. Rosenkrantz, and S. Ravi. Spanning trees short or small. In *Proc. of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 1994.

[30] C. Roberts et al. Signaling and circuitry of multiple MAPK pathways revealed by a matrix of global gene expression profiles. *Science*, 287:873–880, 2000.

[31] J. Scott, T. Ideker, R. M. Karp, and R. Sharan. Efficient algorithms for detecting signaling pathways in protein interaction networks. In *Proc. of RECOMB*, pages 1–13, 2005.

[32] X. Yan and J. Han. gSpan: Graph-based substructure pattern mining. In *ICDM*, 2002.

[33] X. Yan and J. Han. CloseGraph: Mining closed frequent graph patterns. In *KDD*, 2003.