Visualizing Attributed Graphs via Terrain Metaphor

Yang Zhang
Department of Computer Science and Engineering
The Ohio State University
Columbus, Ohio, USA, 43210
zhang.863@osu.edu

Yusu Wang
Department of Computer Science and Engineering
The Ohio State University
Columbus, Ohio, USA, 43210
yusu@cse.ohio-state.edu

Srinivasan Parthasarathy
Department of Computer Science and Engineering
The Ohio State University
Columbus, Ohio, USA, 43210
srini@cse.ohio-state.edu

ABSTRACT

The value proposition of a dataset often resides in the implicit interconnections or explicit relationships (patterns) among individual entities, and is often modeled as a graph. Effective visualization of such graphs can lead to key insights uncovering such value. In this article we propose a visualization method to explore attributed graphs with numerical attributes associated with nodes (or edges). Such numerical attributes can represent raw content information, similarities, or derived information reflecting important network measures such as triangle density and centrality. The proposed visualization strategy seeks to simultaneously uncover the relationship between attribute values and graph topology, and relies on transforming the network to generate a terrain map. A key objective here is to ensure that the terrain map reveals the overall distribution of components-of-interest (e.g. dense subgraphs, k-cores) and the relationships among them while being sensitive to the attribute values over the graph. We also design extensions that can capture the relationship across multiple numerical attributes. We demonstrate the efficacy of our method on several real-world data science tasks while scaling to large graphs with millions of nodes.

1 INTRODUCTION

Our ability to produce and store data has far outstripped our ability to analyze and utilize this data to derive actionable insight. Many phenomena and problems from all walks of human endeavor can often be represented in graph or network form, where nodes represent entities-of-interest and edges represent interactions or relationships among them. Examples abound across the physical, biological, business, technological, sociological and health domains. A fundamental challenge is the ability to visualize such interconnections at scale while working within the pixel limits of modern displays. This in turn has led to several recent advances in the database systems [27, 28, 31, 34], network science[1, 2, 4], geometry[12, 20, 29, 30], and information visualization [8, 11, 15] communities.

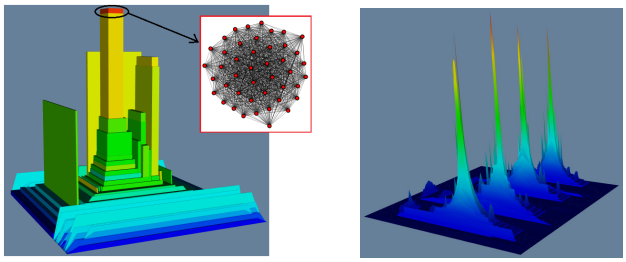
However, the data scientist is often interested in uncovering patterns that go beyond layout and visualization designs – for instance accounting for measures or attributes defined on both nodes and edges of the graph [31, 36]. Such measures can encode explicit information about nodes (properties) and edges (inter-connection properties) but may also encode implicit information about connectivity – both locally (e.g. triangle density, clustering coefficient, K-Core, K-Truss) as well as globally (e.g. closeness, betweenness, PageRank and Influence). Beyond just measures from the topology, such measures may also incorporate heterogeneous information related to content (e.g. sequence information of a protein etc.). Visualizing the measure information in such graphs (where each node or edge has one or more attributes associated with it) further exacerbates the challenge.

In this article we propose a novel visualization method to explore graphs with such *measures* associated with nodes (or edges). Each measure could either be a natural attribute of a heterogeneous graph or a derived attribute (e.g. triangle density, centralities, cliquishness) [27, 34]. Focusing on numerical and ordinal measures-of-interest, one may model the problem as a continuous function $f : X \to \mathcal{R}$: the domain X is a simplicial complex whose vertex set is the set of input graph nodes, and its topology is determined by the input graph topology. We call such a representation of a graph a *graph-induced scalar field*. We then leverage a powerful "terrain metaphor" idea for visualizing such fields [7, 12, 29] and develop novel extensions that can accommodate both node and edge measures-of-interest. Our visualization model *naturally encodes both topological and attributal measure information together*, and is capable of handling large graphs with millions of nodes/edges. Empirically we demonstrate the use of our methodology on a range of data science tasks while providing a comparative assessment against state-of-the-art alternatives from the perspectives of efficacy, efficiency and usability.

Figure 1 previews our methodology for the tasks of: i) visualizing dense subgraphs (K-Cores, K-Trusses, etc) and their hierarchical inter-relationships; and ii) visualizing communities in social networks. In Figure 1(a), we use K-Core number [4] as a measure on each node and use it to visualize K-Cores in a collaboration network (a fundamental network science task), where the top part of high peaks contains dense K-Cores (Clicking on the circled part of the high peak will show the details of a dense K-Core in the red box). In Figure 1(b), we use community score [32] as a measure to visualize the four communities in a DBLP network, where each major peak is a community, and the sub-peaks in a major peak indicate sub-communities. The top part of a peak contains key members of the community. Our proposed tool can also color the terrain using a second measure, so one can analyze the correlation between two different measures on the graph and furthermore allows for various rotation/transformation operators which allow an end-user to interrogate the data from multiple perspectives. Further details of our evaluation is detailed in Section 3. In summary:

KDD'17, August 13–17, 2017, Halifax, NS, Canada.

ISBN 978-1-4503-4887-4/17/08…$15.00
DOI: http://dx.doi.org/10.1145/10.1145/3097983.3098130



(a) Visualizing K-Cores (b) Visualizing Communities

Figure 1: Examples of Terrain Visualization

- We advocate a novel terrain visualization method to visualize a graph whose vertices/edges are associated with measures-of-interest. Our method enables users to quickly examine how such measures are distributed over the graph.
- We propose the concept of maximal α-connected component to represent various graph patterns (dense subgraphs, communities, K-Cores, K-Truss etc.), and show that our visualization method not only captures the distribution of graph patterns, but also their hierarchical relationships.
- Based on our visualization, we propose methods to analyze the relation between multiple measures on one graph.
- Finally, we empirically demonstrate that the visualization method is both general-purpose and scalable and can handle graphs with millions of nodes and edges.

2 VISUALIZING ATTRIBUTED GRAPHS

Notation: We define a vertex-based heterogeneous scalar graph *G(V, E)* as a graph comprising edges E and vertices V, where each vertex v has one scalar value[1] associated with it, denoted as *v.scalar*. In the following we refer to vertex-based scalar graph as a *scalar graph* for expository simplicity. We also assume the following notation in the rest of this section. For two subgraphs $G'(V', E')$ and $G''(V'', E'')$ of scalar graph $G(V, E)$, G' is the **same** as G'' (denoted as $G' = G''$) if $V' = V''$ and $E' = E''$, G' is a **subgraph** of G'' (denoted as $G' \subseteq G''$) if $V' \subseteq V''$ and $E' \subseteq E''$, G' is **connected** to G'' (denoted as $G' \leftrightarrow G''$) if there is a vertex $v' \in V'$ and a vertex $v'' \in V''$ such that $e(v', v'') \subseteq E$. Besides scalar graph, we will also introduce scalar tree in the following sections, and these notations apply analogically to scalar trees.

2.1 Theoretical Insights

We now describe the key insights underpinning our visualization strategy. We first define **maximal α-connected component** as follows:

Definition 2.1. A connected component $C(V_C, E_C)$ of scalar graph $G(V, E)$ is a **maximal α-connected component** if it satisfies following conditions:
(1) for every vertex $v \in V_C$, $v.scalar >= \alpha$.
(2) for any vertex $v \in V_C$, if v' is connected to v and $v' \notin V_C$, then $v'.scalar < \alpha$.
(3) for any edge $e(v_1, v_2) \in E$, if $v_1 \in V_C$ and $v_2 \in V_C$, then $e(v_1, v_2) \in E_C$.

[1]For expository simplicity we assume a single attribute for now. We will eliminate this simplistic assumption in Section 2.6.

Maximal α-connected components are an important aspect of our visualization strategy. We note that the definition accommodates both connectivity (topology) and scalar value information. The scalar values may be inherited from the domain or derived (e.g. local density or k-core values[4])). The distribution of maximal α-connected components in a graph would allow the data scientist to interrogate the distribution and inter-connectivity of such structures within the graph (e.g. distribution and relationships of K-cores).

Definition 2.2. For vertex v, we define **MCC(v)** as the maximal α-connected component containing v where $\alpha = v.scalar$.

Theorem 2.3. *For any maximal α-connected component C in G, there is a vertex v' in G such that $MCC(v') = C$.*

Proof Sketch: From Definitions 2.1 and 2.2 it follows that v' in Theorem 2.3 is the vertex with minimum scalar value in C. The proof trivially follows from this observation. □

Theorem 2.4. *For two vertices v and v', if $v.scalar = v'.scalar$ and $MCC(v)$ contains v', then $MCC(v) = MCC(v')$.*

Proof Sketch: For every vertex v_i in $MCC(v)$, we can show that in $MCC(v)$ there is a path $v_i \rightarrow ... \rightarrow v \rightarrow ... \rightarrow v'$, which starts with v_i, passes through v, and ends at v', and all the vertices on the path have scalar value greater than or equal to $v'.scalar$, so v_i is in $MCC(v')$. Similarly, we can show for every vertex v_j in $MCC(v')$, v_j is in $MCC(v)$. Thus $MCC(v) = MCC(v')$. □

Theorem 2.5. *For a maximal α_1-connected component C_1 and another maximal α_2-connected component C_2, if $C_1 \leftrightarrow C_2$ then $C_1 \subseteq C_2$ or $C_2 \subseteq C_1$.*

Proof Sketch: Based on Theorem 2.3, there are two vertices v_1 and v_2 such that $MCC(v_1) = C_1$ and $MCC(v_2) = C_2$. Assume $v_1.scalar \leq v_2.scalar$ (w.l.g). Since $MCC(v_1) \leftrightarrow MCC(v_2)$, for any vertex v' in $MCC(v_2)$, we can find a path $v_1 \rightarrow ... \rightarrow v'$ connecting v_1 and v', and all the vertices on the path have scalar value greater than or equal to $v_1.scalar$, so v' is in $MCC(v_1)$. Thus $MCC(v_2) \subseteq MCC(v_1)$. □

Theorems 2.3 and 2.4 bound the number of distinct maximal α-connected components that need to be processed by any algorithm (number of vertices in the graph). Theorem 2.5 highlights an important hierarchical relationship between different maximal α-connected components which will be exploited by our visualization strategy. Full proofs can be found in our extended paper[35].

2.2 Vertex Scalar Tree

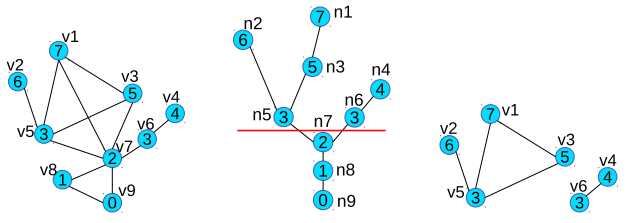
In this section, we describe the vertex scalar tree (*scalar tree* for short) to analyze a scalar graph. We note that if one views the graph as a 1-dimensional simplicial complex, the vertex scalar values induce a piecewise-linear function on this domain, then the maximal α-connected components we defined previously are akin to level sets or contours for each α [6, 30]. We also note that this perspective also helps us easily extend the notion of scalar tree for edge-based scalar graphs (in Section 2.3).

A scalar tree is a tree in which every node is associated with a scalar value, and the scalar tree T of scalar graph G has the following properties:

(1) Every node in T corresponds to a vertex in G with the same scalar value, and vice versa (i.e. one-to-one correspondence).
(2) Every maximal α-connected component in G corresponds to a subtree in T, and vice versa (i.e. one-to-one correspondence).
(3) Assume a maximal α_1-connected component C_1 corresponds to subtree T_1 in T, and another maximal α_2-connected component C_2 corresponds to subtree T_2 in T, then C_1 is a subgraph of C_2 if and only if T_1 is subtree of T_2.
(4) Assume a maximal α_1-connected component C_1 corresponds to subtree T_1 in T, and another maximal α_2-connected component C_2 corresponds to subtree T_2 in T, then C_1 and C_2 are not connected if and only if T_1 and T_2 are not connected.

Notation: In the following text, the scalar tree node corresponding to vertex v is denoted as $n(v)$, and the vertex corresponding to scalar tree node n is denoted as $v(n)$. The parent of tree node n is denoted as $parent(n)$. The subtree corresponding to a maximal α-connected component C is denoted as $ST(C)$. A subtree ST containing nodes $n_1, n_2 ... n_k$ is denoted as $ST(n_1, n_2, ..., n_k)$, and a connected component C containing vertices $v_1, v_2, ... v_k$ is denoted as $C(v_1, v_2, ..., v_k)$.

Example: In Figure 2 we use an example to illustrate the properties of scalar tree. Figure 2(a) is a scalar graph G, in which every vertex's label is its scalar value. Figure 2(b) is a correspondent scalar tree T of Figure 2(a), rooted at node n_9. Node n_i in Figure 2(b) corresponds to vertex v_i in Figure 2(a) (Property 1). In Figure 2(c), we extract all the maximal 2.5-connected components of G:
$C_1(v_1, v_2, v_3, v_5)$ and $C_2(v_4, v_6)$, and in Figure 2(b), their correspondent subtrees are: $ST(C_1) = ST(n_1, n_2, n_3, n_5)$ and $ST(C_2) = ST(n_4, n_6)$, this satisfies Property 2. We notice that C_1 and C_2 are not connected, and $ST(C_1)$ and $ST(C_2)$ are not connected either, this satisfies Property 4. We also observe that C_1 is a subgraph of a maximal 2-connected component $C_3(v_1, v_2, v_3, v_4, v_5, v_6, v_7)$, and $ST(C_1)$ is a subtree of $ST(C_3) = ST(n_1, n_2, n_3, n_4, n_5, n_6, n_7)$, this satisfies Property 3.



(a) Scalar Graph G (b) Scalar Tree T (c) maximal 2.5-connected components

Figure 2: Scalar Graph and Scalar Tree

From the properties above, we can see that scalar tree captures the distribution and relationship of all maximal α-connected components within a scalar graph.

Constructing the Vertex Scalar Tree: Algorithm 1 constructs the scalar tree by leveraging the observation connecting our problem with level sets and contour trees[6, 30]. Algorithm 1 processes all the vertices in decreasing order of scalar values (line 1-3). If the current vertex v_i is connected to a previously processed vertex v_j, but $n(v_i)$ is not in the current tree of $n(v_j)$ (line 4-5), then in line 6 we connect $n(v_i)$ with the root of the current tree of $n(v_j)$ (denoted as $root(n(v_j))$). Now $n(v_i)$ is parent of $root(n(v_j))$, so $n(v_i)$ becomes the new root of the current tree containing $n(v_j)$.

Algorithm 1 Constructing Scalar Tree

Require: A scalar graph G(V, E).
Ensure: The scalar tree T of G.
1: Sort vertices in decreasing order of scalar values, the sorted vertices are $v_1, v_2, ...v_n$;
2: Create a tree node $n(v_i)$ for each vertex v_i;
3: **for** $i = 1$ to n **do**
4: **for** every neighbor v_j of v_i **do**
5: **if** $j < i$ and currently $n(v_i)$ and $n(v_j)$ are not in the same tree **then**
6: Connect $n(v_i)$ to $root(n(v_j))$; //$n(v_i)$ is parent

The running time of line 1 is $O(|V| \cdot log|V|)$. An efficient implementation of line 5 uses the "Union Find" data structure, which compares $root(n(v_i))$ and $root(n(v_j))$. The amortized time for "Union Find" (line 5) is $O(\alpha(n))$ per operation, where $\alpha(n)$ is inverse of Ackermann function, and is usually a small constant. So the total worst-case time cost of Algorithm 1 is $O(|E| \cdot \alpha(n) + |V| \cdot log|V|)$

In the scalar tree T generated by Algorithm 1, every node's scalar value is greater than or equal to its parent's scalar value. If we layout a scalar tree T in such a way that the height of each node is the scalar value of the node, then we could get all the maximal α-connected components for a particular α in a simple way: draw a line with $height = \alpha$ to cross T, and each of the subtrees above the line corresponds to a maximal α-connected component. For example, as Figure 2(b) shows, the two subtrees above red line $height = 2.5$ correspond to all the maximal 2.5-connected components. When every vertex in the input scalar graph G has a distinct scalar value, the scalar tree T generated by Algorithm 1 has the following property:

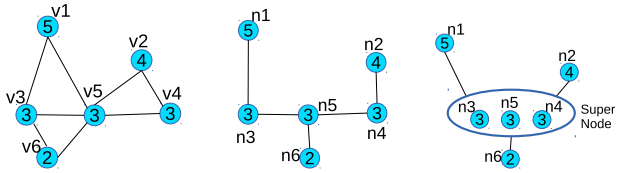
Proposition 2.6. *For every vertex v in G, assume it corresponds to n(v) in T, then the subtree rooted at n(v) (denoted as ST) in T corresponds to the MCC(v).*

Proof. Obviously ST corresponds to a connected component SG. Since $n(v).scalar$ is the minimum scalar value in ST, every vertex in SG has scalar value greater than or equal to $v.scalar$. If there is a vertex v_i that connects to a vertex v_j in SG, and $v_i.scalar > v.scalar$, then due to Algorithm 1, $n(v_i)$ and $n(v_j)$ will be in the same subtree ST, which indicates that v_i is in SG. So SG is a maximal $v.scalar$-connected component, and it is MCC(v). □

Based on Theorem 2.3, Theorem 2.5 and Proposition 2.6, it follows that when every vertex in the input scalar graph has a *distinct value*, the tree generated by Algorithm 1 has the four properties of the scalar tree. We do still have to handle the following corner case - when some vertices in scalar graph have the same scalar value, we need to do some postprocessing, described next.

Postprocessing the Vertex Scalar Tree: If some vertices in G have the same scalar values, the scalar tree generated by Algorithm 1 may not satisfy Property 2 – a subtree may not correspond to a maximal α-connected component. For example, Figure 3(a) is a scalar graph, and Figure 3(b) is the scalar tree generated by

Algorithm 1. The subtree rooted at n_3 is $ST(n_1, n_3)$ corresponds to connected component $C(v_1, v_3)$, however, $C(v_1, v_3)$ is not a maximal α-connected component.



(a) Scalar Graph G (b) Scalar Tree T (c) Super Scalar Tree

Figure 3: Postprocessing Scalar Tree

To solve the problem, we need to postprocess the scalar tree T generated by Algorithm 1: we merge the ancestor node with all its descendants with the same scalar value into a super node. The correctness of the algorithm is based on the following proposition:

Proposition 2.7. *For any tree node n in T, assume it has an ancestor n_{anc} in T, such that $n_{anc}.scalar = n.scalar$, and $parent(n_{anc})$ is null or $parent(n_{anc}).scalar < n_{anc}.scalar$, then the subtree rooted at n_{anc} (denoted as ST_{anc}) corresponds to the MCC(v(n)).*

Proof. Obviously the subtree rooted at n_{anc} is MCC($v(n_{anc})$). Since subtree ST_{anc} contains node n, MCC($v(n_{anc})$) contains vertex $v(n)$. Because $n_{anc}.scalar = n.scalar$, from Theorem 2.4, we can get that $MCC(v(n_{anc})) = MCC(v(n))$. The proposition is true. □

In the example in Figure 3(b), we will merge the nodes n_3, n_4, n_5 into a super node, and build a super tree (Figure 3(c)). Every subtree of the super tree will correspond to a maximal α-connected component. After postprocessing, the scalar tree will still satisfy Property 2, 3, 4, but may not satisfy Property 1, because a super node may correspond to multiple vertices, however, this does not affect the further analysis. The postprocessing only needs one pass of the scalar tree T, so the time complexity is $O(|V|)$.

2.3 Edge-based Scalar Graph

Here we describe a *novel* approach for modeling scalar values on edges. We define an edge-based scalar graph $G(V, E)$ as a graph comprising edges E and vertices V, where each edge e has one scalar value associated with it, denoted as *e.scalar*. Similarly, the **maximal α-edge connected component** is defined below:

Definition 2.8. A connected component $C(V_C, E_C)$ of scalar graph $G(V, E)$ is a **maximal α-edge connected component** if it satisfies following conditions:
(1) for every edge $e \in E_C$, $e.scalar >= \alpha$.
(2) for any edge $e \in E_C$, if edge e' shares a common vertex with e and $e' \notin E_C$, then $e'.scalar < \alpha$.
(3) for any edge $e(v_1, v_2) \in E_C$, we have $v_1 \in V_C$ and $v_2 \in V_C$.

The edge scalar tree T of edge scalar graph G has the following properties:

(1) Every node in T corresponds to an edge in G with the same scalar value, and vice versa (i.e. one-to-one correspondence).
(2) Every maximal α-edge connected component in G corresponds to a subtree in T, and vice versa (i.e. one-to-one correspondence).
(3) Assume a maximal α_1-edge connected component C_1 corresponds to subtree T_1 in T, and another maximal α_2-edge connected component C_2 corresponds to subtree T_2 in T, then C_1 is a subgraph of C_2 if and only if T_1 is subtree of T_2.
(4) Assume a maximal α_1-edge connected component C_1 corresponds to subtree T_1 in T, and another maximal α_2-edge connected component C_2 corresponds to subtree T_2 in T, then C_1 and C_2 are not connected if and only if T_1 and T_2 are not connected.

Naive Method: a naive way to build edge scalar tree is to first convert the edge-based scalar graph $G(V, E)$ to a dual graph $G_d(V_d, E_d)$– where every edge in G is converted to be a vertex in G_d, and if two edges in G share a common vertex, their correspondent vertices in G_d are connected. We then apply Algorithm 1 to G_d – the generated tree is an edge scalar tree of G. The time complexity of the method is $O(|E_d| \cdot log|V_d| + |V_d| \cdot log|V_d|)$. In the dual graph G_d, we have $|V_d| = |E|$ and $|E_d| = O(\sum_{v \in V} degree(v)^2)$, so the time complexity is actually $O(\sum_{v \in V} degree(v)^2 \cdot log|E| + |E| \cdot log|E|)$. The time cost is high because the bottleneck $\sum_{v \in V} degree(v)^2$ could be $|V|^3$ in the worst case.

Optimized Method: We propose a novel, more efficient method (Algorithm 2) to construct edge scalar tree from the edge scalar graph, and the time complexity is reduced to be $O(|E| \cdot log|E|)$. In line 1, we sorted all the edges in descending order of scalar value. In line 2-3, we select the *min_id_edge* on vertex v that has the minimum index. In line 6-9, we process edge e_i. Instead of checking all e_i's neighbor edges (edges which share a common vertex with e_i) , we just need to check the *min_id_edge*s of e_i's two vertices (line 6-7). This is based on Proposition 2.9:

Proposition 2.9. *If edge e_j is a neighbor edge of e_i ($i > j$), and they share the same vertex v, when processing e_i in line 6-9 of Algorithm 2, root($n(e_j)$) is the same as root($n(v.min\_id\_edge)$).*

Proof. Since $i > j$, e_j will be processed before e_i. When processing e_j in line 6-9 of Algorithm 2, $n(e_j)$ will be connected to root($n(v.min\_id\_edge)$), which means $n(e_j)$ and $n(v.min\_id\_edge)$ will be in the same tree thereafter. So when processing e_i, root($n(e_j)$) is the same as root($n(v.min\_id\_edge)$). □

Algorithm 2 Constructing Edge Scalar Tree

Require: An edge scalar graph G(V, E).
Ensure: The edge scalar tree T of G.
1: Sort edges in decreasing order of scalar values, the sorted edges are $e_1, e_2, ...e_n$.
2: **for** each vertex v in G **do**
3: set $v.min\_id\_edge$ to be the edge on v that has the minimum index.
4: Create a tree node $n(e)$ for each edge e.
5: **for** $i = 1$ to n **do**
6: Assume e_i has two vertices v_1 and v_2, create an array: $min\_neighbors[2] = \{v_1.min\_id\_edge, v_2.min\_id\_edge\}$;
7: **for** each edge e_m in $min\_neighbors$ **do**
8: **if** $m < i$ and currently $n(e_i)$ and $n(e_m)$ are not in the same tree **then**
9: connect $n(e_i)$ to root($n(e_m)$) //$n(e_i)$ is parent

The time complexity of line 1 in Algorithm 2 is $O(|E| \cdot log|E|)$. For each edge e, line 8 is executed $O(1)$ times, so line 8 is executed a total of $O(|E|)$ times, and the total running time of line 5-9 is $O(|E| \cdot log|E|$. The worst case running time of Algorithm 2 is $O(|E| \cdot log|E|)$.

2.4 Relationship between maximal α-(edge) connected component and Dense Subgraph

A dense subgraph is a connected subgraph in which every vertex is heavily connected to other vertices in the subgraph. K-Core [4] and K-Truss [14, 26](also called Triangle K-Core in [34], DN-graph in [28]) are two common dense subgraph patterns that draw much attention in recent works. The definitions of K-Core and K-Truss are as follows:

Definition 2.10. A K-Core is a subgraph in which each vertex participates in at least K edges within the subgraph. The maximal K-Core of a vertex v is the K-Core containing v that has the maximum K value, and the K value of maximal K-Core of v is denoted as KC(v).

Definition 2.11. A K-Truss is a subgraph in which each edge participates in at least K triangles within the subgraph. The maximal K-Truss of an edge e is the K-Truss containing e that has the maximum K value, and the K value of maximal K-Truss of e is denoted as KT(e).

Now we prove the relationship between maximal α-(edge) connected component and K-Core, K-Truss below.

Proposition 2.12. *In a scalar graph G where for any vertex* v, v.scalar = KC(v)*, a maximal α-connected component in G is a K-Core where $K = \alpha$.*

Proof. Assume in a maximal α-connected component C, vertex v has the minimum scalar value. Based on the definition of K-Core, for every vertex v' in the maximal K-Core of v, $KC(v') >= KC(v)$, so the maximal K-Core of v is a subgraph of C. Since v is connected to at least $KC(v)$ vertices in its maximal K-Core, v is connected to at least $KC(v)$ vertices in C. For every other vertex v' in C, similarly we can get that v' is also connected to at least $KC(v)$ vertices in C. So C is a K-Core where $K = KC(v)$, since $KC(v) = v.scalar >= \alpha$, C is also a K-Core where $K = \alpha$. □

Proposition 2.13. *In an edge scalar graph G where for any edge* e*, $e.scalar = KT(e)$, a maximal α-edge connected component in G is a K-Truss where $K = \alpha$.*

The proof is similar to the proof of Proposition 2.12 and is omitted in the interests of space. Note that when we define the scalar value of each vertex/edge to be KC(v)/KT(e), the (edge) scalar tree will capture the distribution and relationships among K-Cores or K-Trusses in the graph.

2.5 Visualization via Terrain Metaphor

Scalar trees are usually not easy to visually interpret, especially when the size of the tree is too large. We adapt the terrain metaphor – topological landscape visualization technique defined on scalar-valued functions [12] – to visualize scalar graphs.

In Figure 4 we use an example to illustrate how to convert the scalar tree in Figure 4(a) to terrain visualization in Figure 4(c). In Figure 4(b) we first layout all the tree nodes of Figure 4(a) in a 2D plane, every node n_i is represented by a boundary b_i in the 2D plane, and the area enclosed by the boundary b_i is proportional to the number of nodes in subtree (not including n_i) rooted at n_i. To generate the 2D layout, we start traversing the tree from the root(bottom) node n_9, draw the outermost boundary b_9 to represent it. Then we move to n_8, and draw a boundary b_8 inside b_9. When we reach n_7, and draw boundary b_7, we find there are two subtrees rooted at n_7, so we split the area inside b_7 into 2 areas, and recursively layout each subtree in each area. When we reach leaf nodes n_1, n_2, n_4, since the size of their subtrees is 0, their correspondent boundaries degenerate to be points.

To convert the 2D layout (Figure 4(b)) into a terrain visualization in 3D space (Figure 4(c)), we first escalate each boundary b_i in Figure 4(b) to the height of $n_i.scalar$, and then draw a "wall" between neighboring boundaries. Finally we generate a terrain in Figure 4(c). We can color the terrain by assigning each vertex a color value, and since each "wall" is confined by two boundaries b_i and b_j, we color the wall based on the color value of the vertex corresponding to b_i or b_j. [2]

To identify a subtree of the scalar tree in the terrain visualization, we locate the correspondent boundary b_r of the subtree root n_r, and the terrain area within the boundary b_r corresponds to the subtree. In our paper, we define a $peak_\alpha$ in terrain as below:

Definition 2.14. A $peak_\alpha$ is the terrain area within a boundary whose height is α.

Since each $peak_\alpha$ corresponds to a subtree in scalar tree, we can easily get the following properties of a $peak_\alpha$:

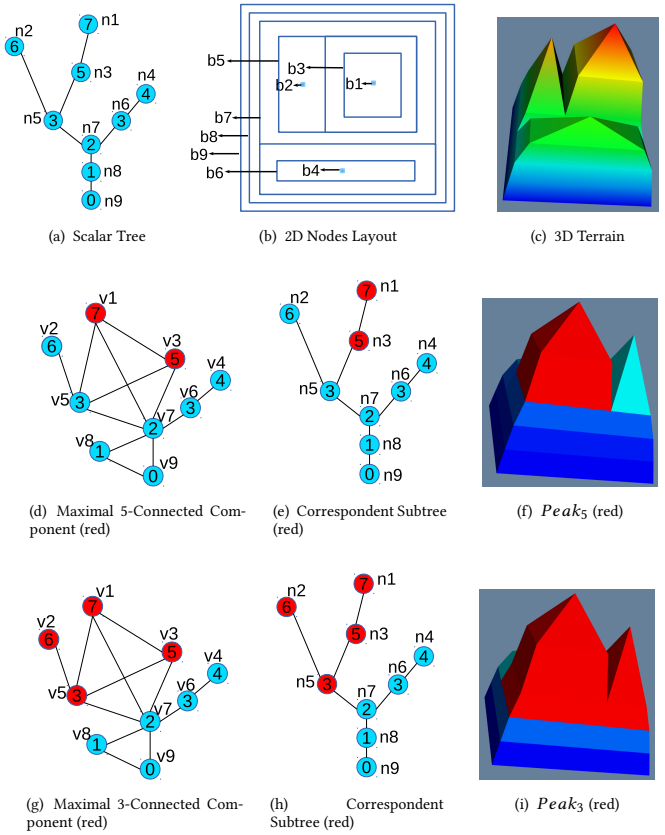
(1) A $peak_\alpha$ corresponds to a maximal α-connected component (denoted as $C(peak_\alpha)$) in the scalar graph and vice versa.
(2) A $peak_\alpha$ is contained in another $peak_{\alpha'}$ if and only if $C(peak_\alpha)$ is a subgraph of $C(peak_{\alpha'})$.
(3) A $peak_\alpha$ and another $peak_{\alpha'}$ are not connected if and only if $C(peak_\alpha)$ and $C(peak_{\alpha'})$ are not connected.

For example, the red peak in Figure 4(f) is a $peak_5$, which corresponds to the maximal 5-connected component (red nodes) in Figure 4(d), and the red peak in Figure 4(i) is a $peak_3$, which corresponds to the maximal 3-connected component (red nodes) in Figure 4(g). One $peak_\alpha$ may contain some sub-peaks, which indicates its maximal α-connected component contains other maximal α'-connected components. For example, $peak_5$ in Figure 4(f) is contained in $peak_3$ in Figure 4(i), this indicates that the correspondent maximal 5-connected component in Figure 4(d) is a subgraph of the maximal 3-connected component in Figure 4(g). In a $peak_\alpha$, the area of its bottom boundary indicates the number of vertices in its correspondent maximal α-connected component.

User Interaction: Our terrain visualization tool, provides following features to help users interact with the terrain.

Rotate: the user could rotate the terrain to look at it from different angles. For example, Figure 4(c) and Figure 4(f) show the same terrain from two different viewpoints.

[2] The use of 3D rather than 2D was a conscious one. First we found that the 3D abstraction better matched users' mental-map of the "terrain" concept as well as the hierarchical relationships amongst components-of-interest. While 3D-layouts can pose a problem with respect to occlusion, the ability to interactively rotate the point-of-view along with the ability to link 2D-layouts of regions-of-interest alleviates this issue.



(a) Scalar Tree (b) 2D Nodes Layout (c) 3D Terrain

(d) Maximal 5-Connected Component (red) (e) Correspondent Subtree (red) (f) $Peak_5$ (red)

(g) Maximal 3-Connected Component (red) (h) Correspondent Subtree (red) (i) $Peak_3$ (red)

Figure 4: Terrain Visualization of a Simple Scalar Tree

Zoom in/out: the user can zoom in/out to see the details/overview of the terrain. For example, in Figure 7(a), we zoom into the terrain in the left picture, and get a clear picture of the two peaks in the right picture.

Linked-2D-Displays: Our tool allows the user to select any region of the terrain, and invoke a "callback" function to visualize the selected region using other visualization methods. For example, in Figure 5(c), we select the region in the white dashed line box, and 2D-linked spring layout visualization method to draw the selected region in the red box beside it. Expert users optionally can use the "callback" function to integrate our terrain visualization with other customized visualization methods.

2.6 Generalizations

Multiple Scalar Fields: In situations where multiple attributes are defined on each vertex or edge of a heterogenous graph, one may be interested in how they correlate with each other and the graph topology. One approach is to reduce the multiple scalar fields to a low dimensional embedding (e.g. via multidimensional scaling), and subsequently use our following methods to analyze and visualize a few scalar fields. Our methods can also be used for a graph without attributes - we can convert it to attributed graph by representing each vertex in a low dimensional space through network embedding [22].

Local Correlation Index: Assume we have two scalar fields (or a two dimensional embedding), S_i and S_j, defined on a graph. Each vertex v has scalar values $v.scalar_i$ and $v.scalar_j$ in the two scalar fields. Informally, we define the Local Correlation Index to measure the correlation of two scalar fields on local areas of a graph. Here local area is defined as local neighborhood (e.g. 1-hop) of each vertex v(denoted as N(v)). The Local Correlation Index of S_i and S_j on N(v) is denoted as $LCI_{Si,Sj}(v)$. For each vertex v, we compute $LCI_{Si,Sj}(v)$ as follows.

$$\overline{v.scalar_i} = \frac{\sum_{u \in N(v)} u.scalar_i}{|N(v)|}$$

$$Cov_{ij}(v) = \frac{\sum_{u \in N(v)} (u.scalar_i - \overline{v.scalar_i}) * (u.scalar_j - \overline{v.scalar_j})}{|N(v)|}$$

$$LCI_{Si,Sj}(v) = \frac{Cov_{ij}(v)}{\sqrt{Cov_{ii}(v)} * \sqrt{Cov_{jj}(v)}}$$

$LCI_{Si,Sj}(v)$ is actually the correlation of the scalar values of S_i and S_j on v's k-hop neighborhood. This measure is easily extensible to analyze edge-based scalar graphs.

Global Correlation Index: We can compute the Global Correlation Index (GCI) of scalar fields S_i and S_j on a graph by averaging the Local Correlation Indexes of all neighborhoods-of-interest (entire graph or a region-of-interest within the graph selected by our tool).

$$GCI_{S_i,S_j}(G) = \sum_{v \in V} LCI_{S_i,S_j}(v)/|V|$$

As we shall demonstrate in the empirical analysis, comparing the Global Correlation Index and Local Correlation Index, can yield useful visual insights on outlier neighborhoods within such graphs.

Visualization Strategies: To visualize the local correlation between two scalar fields, we can use $LCI_{Si,Sj}(v)$ as a scalar field to draw the terrain. This will show us the overall distribution of $LCI_{Si,Sj}(v)$ over the graph, and help us identify the area of the graph where the two scalar fields are positively/negatively correlated.

We can also visually capture the global correlation of scalar fields S_i and S_j through coloring terrain visualization. We use one scalar field S_i to draw the terrain visualization, and use the other scalar field S_j to color the terrain (see Figure 1(a)). We also note that this methodology *can also be extended visualize the relationship between a numerical or ordinal attribute and nominal attributes of vertices*, by coloring the terrain based on the value of nominal attributes.

2.7 Related Context

We are now in a position to briefly place the proposed visualization strategy in the context of related work. Visualizing graph data is an important problem (See [25] for a recent survey – due to interests of space here we focus only on the most relevant). Gronemann et al. [10] (similarly Athenstadt et al. [3]) use topographic maps and leverage GIS tools to visualize clustering structure within a graph – each mountain corresponds to a cluster. van Liere et al.[24] propose the GraphSplatting method to visualize a graph as a 2D splat field, and use color to encode vertex density. Telea et al.[23] generate a concise representation of graph by bundling similar edges together. While effective on small scale datasets for displaying overall cluster structure, a limitation of these approaches is that it is hard to glean the hierarchical relationships among components-of-interest. Additionally, these methods simply do not scale to

large data with millions of edges nor do they account for attributed graphs (scalar values). Bezerianos *et al* [5], Yang *et al*[33], and Shi *et al* [21] propose interactive visual systems to explore networks with node or edge attributes. However, these methods do not consider hierarchical relationships among components-of-interest and graph attributes simultaneously. Sariyuce et al. [19] proposed (r, s)-nucleus to denote a dense subgraph, and use a forest of nuclei to represent hierarchical structure of a graph. The difference is, their method focuses on density of graph topology, while our method focuses on relation between scalar values and graph topology. Also, they do not explicitly consider visualization, in their work.

In summary, a major difference between our work and prior art is that we propose to analyze the graph through the hierarchical structure (scalar tree) induced by the ***maximal α-connected components***. The benefit is that it *naturally* encodes the relationship between clustering structure and scalar values – it highlights how scalar values evolve from high values to low values over the graph. This is particularly useful for a data scientist who wishes to understand how a community is expanded from its core members to peripheral members(Figure 7).Furthermore, based on different attributes, the ***maximal α-connected component*** can represent different subgraph patterns, such as K-core, K-truss, subcommunity, which has attracted much interest within the database community [4, 26, 27], to reveal the topological relationship (containment, connection) among components of interest (e.g. K-Cores, K-Trusses, communities). We examine these issues next.

3 EXPERIMENTAL EVALUATION

We seek to evaluate the effectiveness (qualitative) and efficiency of our interactive network visualization method in this section. We leverage a wide range of datasets from the network science community (some are from [16]) as noted in Table 1. Heights in our terrain visualization represent scalar measures of input scalar graph while color represents intensity of the same measure (unless otherwise noted). The color ranges from red (most intense); yellow (intense); green (less intense); blue (least intense). All experiments are evaluated on a 3.4GHz CPU, 16G RAM Linux-based desktop. The visualization tool is built on top of the Denali platform[7]. Additional results (including a detailed user study) can be found in our extended paper [35].

Table 1: Dataset Properties

Dataset	# Nodes	# Edges	Context
GrQc	5242	14496	Coauthorship in General Relativity and Quantum Cosmology
Wikivote	7115	103689	Who-votes-who in Wikipedia
Wikipedia	1,815,914	34,022,831	Links between Wikipedia pages
PPI	4741	15147	Protein Protein Interaction network
Cit-Patent	3,774,768	16,518,947	Citations made by patents
Amazon	334863	925872	Co-Purchase relationship
Astro	17903	196972	Coauthorship in Astro Physics
DBLP	27199	66832	Coauthorship in Computer Science

3.1 Visualizing Dense Subgraphs

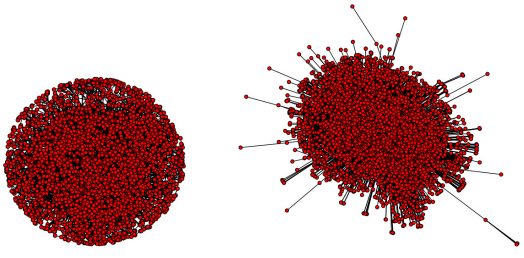
Effectiveness: The visualization of dense subgraphs within graphs has been of much interest within the database and information visualization. Examples abound and include CSV plots [27], K-Core [1] and Triangle K-Core (K-Truss) [34] plots. Here we use our terrain visualization to visualize K-Cores and K-Trusses and compare with the previous methods.

We consider two datasets (GrQc, Wikivote) for this illustration. In Figure 5(a) and Figure 5(b), we use the traditional spring layout Algorithm [9] to draw both networks – it is hard to say anything about the distribution of dense subgraphs using such a plot. Following the discussion in Section 2.4, we use KC(v) as scalar value, and generate the terrain visualization of both networks in Figure 5(c) and Figure 5(d). Recall that if the scalar value of every vertex v is defined to be KC(v), each maximal α-connected component is a K-Core where $K = \alpha$. Thus in the terrain, each $peak_\alpha$ is a K-Core where $K = \alpha$. The distribution of K-Cores in the two datasets is obviously different. Figure 5(d) shows that there is one single high peak, which means the network has one densest K-Core, and K-Core density gradually decreases to the neighboring vertices. Figure 5(c) shows that there are several high peaks within the GrQc network, which means there are several disconnected K-Cores with high K values (dense K-Cores).

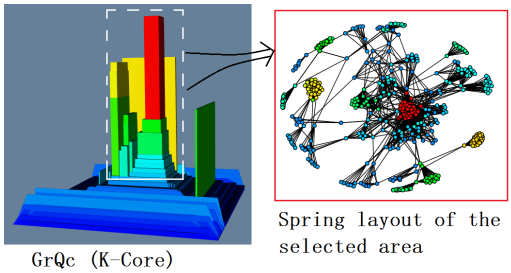
Moreover, Figure 5(c) clearly illustrates the hierarchical relationship among K-Cores. In the selected terrain area (the terrain area in dashed line), the red peak is placed on green and blue foundation, which means the dense K-Core is contained in some less dense K-Cores. This can be verified by drawing spring layout of the selected region in the red box (with our tool, a user can select a region, and call other visualization methods to draw the selected region), the red dense K-Core is surrounded by some green and blue vertices. The visualization of hierarchy is important, as it allows an analyst to derive high level insights on the connectivity that is not immediately obvious even in state of the art K-Core plots as shown in Figure 5(f) [1] for the GrQc network.

We can illustrate the same principle when visualizing K-Trusses (used to understand triangle density) instead of K-Cores. Here each edge uses KT(e) as the scalar measure, and we use the edge-based scalar graph for visualizing the K-Trusses in GrQc dataset. The terrain visualization is in Figure 5(e) where high peaks indicate dense K-Trusses. To contrast, Figure 5(g) depicts a CSV plot, a state-of-the-art density plot leveraged within the database community [27, 34]. Again such visualization strategies do not reveal important hierarchical relationships (e.g. contains) among different K-Trusses. Also we note that our visualization method is a common and flexible framework which can render plots based on different scalar measures, and the ability to rotate, filter and extract details on demand (allowing the analyst to quickly identify regions of interest) will help users understand the graph data better. We also conducted a user-study to evaluate the efficacy of our visualization strategy (omitted due to space constraints, see [35]). We find that our approach outperforms existing alternatives (e.g. LaNet-vi [1], OpenOrd [17]) for the interactive, exploratory analysis of such networks [35].

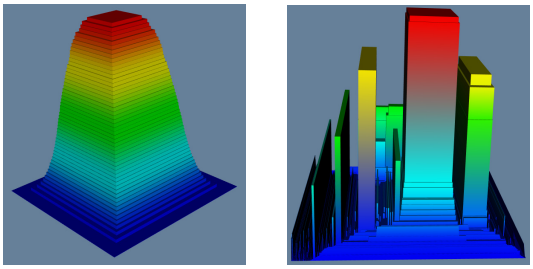
Scalability: We next examine the efficiency of Algorithm 1 and 2. Every dataset has duplicate scalar values, so the generated trees are all super trees. We test our methods on datasets of various sizes, and list the number of nodes in the final super (edge) scalar tree(N_t), time cost to construct the tree (tc) and visualize the tree (tv) in Table 2. The time cost to construct the tree (tc) includes the time cost to construct the tree (Algorithm 1 or 2) and postprocess the tree. The time cost to visualize the tree (tv) is the time cost for the visualization software to read the scalar tree and render the terrain visualization.



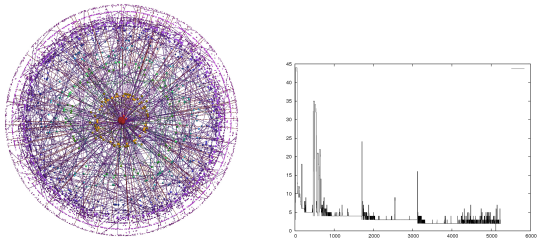
(a) GrQc (spring layout) (b) wikiVote (spring layout)



(c) GrQc (K-Core)



(d) wikiVote (K-Core) (e) GrQc (K-Truss)



(f) 2D Visualization of K-Cores in GrQc (g) 2D plot of K-Trusses in GrQc

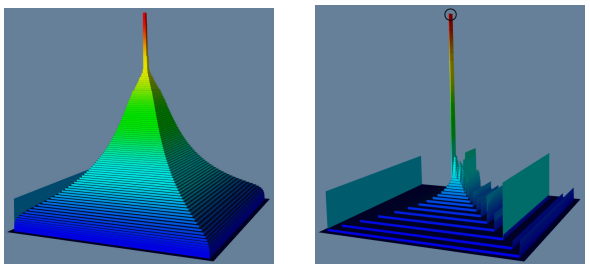
Figure 5: Visualizing Dense Subgraphs in the Network

Table 2: Terrain Visualization Time Cost(sec)

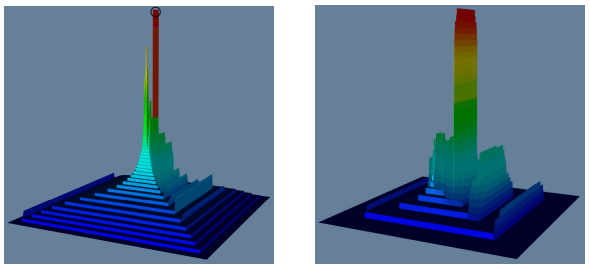
Dataset	Scalar	N_t	tc	te	tv
GrQc	KC(v)	869	0.0018		<1
GrQc	KT(e)	728	0.0039	0.0072	<1
WikiVote	KC(v)	106	0.0037		<1
WikiVote	KT(e)	44	0.053	0.69	<1
Wikipedia	KC(v)	230	6.9		2
Wikipedia	KT(e)	1,903	49.3	16334	22
Cit-Patent	KC(v)	1,059	7.1		2
Cit-Patent	KT(e)	110,412	27.7	65.3	13

Also we list the time cost of the naive method (using dual graph) to build edge-scalar tree (te) in Table 2. We can see the improved method (tc) is much faster than the naive method (te), especially on the Wikipedia dataset, the improved method is more than 300 times faster than the naive method.

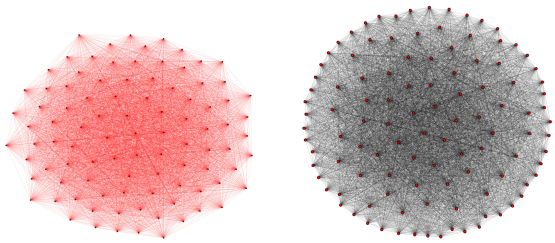
Additionally, in Figure 6, we show the terrain visualizations of (edge) scalar tree of wikipedia and cit-Patent datasets. The peaks in Figure 6(a) 6(b) 6(c) 6(d) indicate dense K-Cores and K-Trusses in the network, we highlight the highest peaks in Figure 6(b) and Figure 6(c), and draw the details in Figure 6(e) and Figure 6(f). They are a K-Truss with K = 86 and a K-Core with K = 64.



(a) Wikipedia Network (K-Core) (b) Wikipedia Network (K-Truss)



(c) Cit-Patent (K-Core) (d) Cit-Patent (K-Truss)



(e) Densest K-Truss of Wiki (f) Densest K-Core of Cit

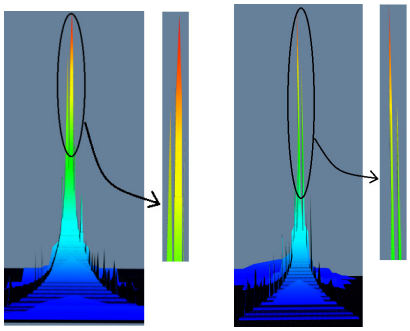
Figure 6: Visualizing K-Cores and K-Trusses

3.2 Visualizing Communities and Roles

Visualizing Community Affiliation: we next illustrate the flexibility of the terrain visualization scheme on an important network science task – understanding community structures (see Figure 7). We use a subset of the DBLP network (DBLP(sub)) for this purpose, comprising authors who publish in the areas of Machine Learning, Data mining, Databases and Information Retrieval. We apply a state-of-the-art overlapping (soft) community detection algorithm [32] on this dataset to detect four communities. Each author in the dataset is affiliated with a community score vector (c_0, c_1, c_2, c_3) indicating how much it belongs to each community. To visualize the affiliation of a particular community i, we use c_i as the corresponding scalar measure, and draw the terrain of the

network. $peak_\alpha$ in the terrain indicates a connected component in which every vertex has $c_i \geq \alpha$.

In Figure 7(a), we visualize community 1, in which most authors are database researchers. We highlight two peaks in the circle of Figure 7(a), and zoom in to get a clear picture of the two peaks on the right. Our tool allows us to easily select authors (vertices) in each peak. We find that authors in the left peak include researchers *Donald Kossmann, Divyakant Agrawal, Amr El Abbadi, Michael Stonebraker, Samuel Madden, and Joseph M. Hellerstein* while authors in the right peak include *Zheng Chen, Hongjun Lu, Jeffrey Xu Yu, Beng Chin Ooi, Kian-Lee Tan, Qiang Yang and Aoying Zhou.* Since authors in both peaks have high community scores (c_1), they can be seen as core members of the community although from different geographic areas. The fact that they are in two separate peaks indicates that authors in one peak do not work with authors in the other peak in the dataset. Similarly, we also observe subcommunities in another community (Figure 7(b)) largely comprising Machine Learning researchers. We also find two peaks in the terrain, and authors in the left peak are *Philip S. Yu, Christos Faloutsos, Michael I. Jordan, Stuart J. Russell, Daphne Koller, Sebastian Thrun, Wei Fan and Andrew Y. Ng,* who all work in United States, while authors in the right peak are *Hang Li, Ji-Rong Wen, Tie-Yan Liu, Lei Zhang, Wei-Ying Ma, Qiang Yang and Yong Yu,* who are researchers in China. Figure 1(b) visualizes the four communities together to give an overview of them.

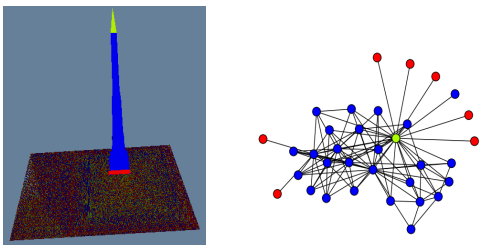


(a) Community 1 (b) Community 2

Figure 7: Visualizing two communities in DBLP network

Visualizing Role and Community Affiliation: Moving beyond community affiliation, the ability to uncover the roles of individual nodes (e.g. bridge, hub, periphery and whisker) within a network or community has received recent interest [13, 18]. Here we examine how one may use the terrain to visualize the distribution of roles over a community. We leverage a recent idea to simultaneously detect communities and roles on large scale networks [18]. For each vertex in the network the algorithm outputs a community affinity vector $(c_1, \ldots, c_m)$ and a role affinity vector $(r_1, \ldots, r_n)$.

As before we focus on a particular community (community i) and use the community score (c_i) of each vertex to create terrain visualization (height). The peak in Figure 8(a) contains the vertices affiliated with one major community in Amazon co-purchase network. Instead of re-using the intensity of community score (c_i) to color vertices we actually use the dominant role for each vertex (four roles is typical [13]) to color vertices as follows: the "hub vertex" is green, the "dense community vertex" is blue, the "periphery vertex" is red, and the "whisker vertex" is white. (see Figure 8(a)). From the terrain visualization, we can see that the vertices in the community have 3 roles, the hub vertex has the highest community score (green top), and below it is the blue portion, which means the "hub vertex" is surrounded by some "dense community vertices" in the network. The red part of the peak indicates that there are some "peripheral vertices" attached to the community. Since the community contains a small number of vertices, we can draw the details of the community using node-link visualization in Figure 8(b). We also note that the roles of nodes not within the particular community are spread out to form a mosaic like structure which is along expected lines [18].



(a) Roles on a community (b) Detail of the community

Figure 8: One community of Amazon co-purchase network

All nodes in Figure 8(b) are books on Amazon, The green node is the book which has the highest salesrank and is focused on creativity (hub). Most of the blue nodes are books about creativity (densely connected), while the red nodes are books loosely relevant to creativity (periphery).

3.3 Comparing Different Centralities

In this section we examine the use of our approach for understanding the relationship of two different centralities across various nodes within a network. We will compare two centralities, degree centrality and betweenness centrality, as two scalar fields, S_d and S_b.

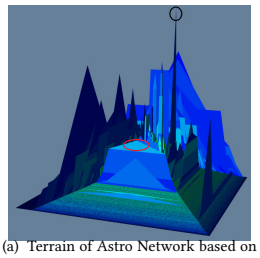
We use the Astro Physics collaboration network, in which each author is a vertex, and each edge indicates a coauthorship between two authors. We first compute the Local Correlation Index of each vertex (as described earlier), and then compute the Global Correlation Index of the network, $GCI_{S_d,S_b} = 0.89$. This indicates that the overall correlation between degree centrality and betweenness centrality is highly positive.

In this case, we are interested in those vertices with negative LCI values, as they could be seen as outliers. We define an outlier score for each vertex v as follows:

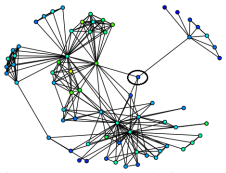
$$outlier\_score(v) = -LCI(v)$$

the vertex with the most negative LCI(v) will have the highest outlier score. We use $outlier\_score(v)$ as scalar field to draw the terrain in Figure 9(a), and we color the terrain using S_d (degree centrality), where red/yellow/blue indicates high/moderate/low degree. We notice that most high peaks (outliers) are blue, which indicates that the outlier vertices usually have low degree.

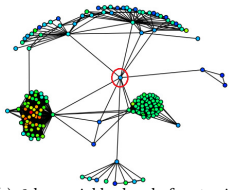
We drill down into the two peaks in the terrain within black and red circles, and select the vertex at the top of each peak. Our software allows us to integrate a different visualization method to inspect the selected two vertices. In this case, we use spring



(a) Terrain of Astro Network based on Outlierness



(b) 2-hop neighborhood of vertex in black circle in Figure 9(a)



(c) 2-hop neighborhood of vertex in red circle in Figure 9(a)

Figure 9: Compare Degree and Betweenness Centralities

layout to draw the two vertices' 2-hop neighborhoods in Figure 9(b) and Figure 9(c). We pick these two vertices specifically because one is in high peak while the other appears to be in a broader but smaller peak. In both cases the vertices picked have high outlier score, which indicates the correlation between degree centrality and betweenness centrality is negative in their neighborhood. Actually the two vertices have relatively higher betweenness and lower degree centrality when compared to many of their neighbors. From Figure 9(b) and Figure 9(c) we can see the two vertices (in the circles) are like bridge nodes connecting multiple communities.

4 CONCLUSIONS

We propose a visualization method to analyze heterogeneous attributed graphs. We demonstrate that our visualization method can reveal important information such as the overall distribution of attribute values over a graph, while simultaneously highlighting components-of-interest (dense subgraphs, social communities, etc.). The method can be extended to analyze relationship between multiple attributes. We evaluate our system on a range of real-world tasks and demonstrated its effectiveness and scalability. In the future we would like to improve the efficiency of our strategy even further by leveraging architecture-conscious designs to scale to larger graphs.

Acknowledgements: This work is supported by NSF grants DMS-1418265, IIS-1550302, CCF-1319406, CCF-1547357 and CCF-1629548. All content reflects the opinion of the authors and may not be shared by their respective employers or sponsors. We also thank Justin Eldridge with help with the Denali platform.

REFERENCES

[1] I. Alvarez-Hamelin, L. Dall'Asta, A. Barrat, and A. Vespignani. 2005. k-core decomposition: A tool for the analysis of large scale Internet graphs. *arXiv:cs.NI/0511007* (2005).
[2] Ana Paula Appel, Ravi Kumar, Deepayan Chakrabarti, Jure Leskovec, Christos Faloutsos, and Andrew Tomkins. 2009. ShatterPlots: Fast Tools for Mining Large Graphs. *SDM* (2009).
[3] Jan Christoph Athenstadt, Robert Gorke, Marcus Krug, and Martin Nollenburg. 2012. Visualizing Large Hierarchically Clustered Graphs with a Landscape Metaphor. *Graph Drawing* (2012).
[4] Vladimir Batagelj and Matjaz Zaversnik. 2003. An O(m) Algorithm for Cores Decomposition of Networks. *CoRR, arXiv.org/cs.DS/0310049* (2003).
[5] A. Bezerianos, F. Chevalier, P. Dragicevic, N. Elmqvist, and J. D. Fekete. 2010. Graphdice: A System for Exploring Multivariate Social Networks. *Proceedings of the 12th Eurographics / IEEE - VGTC conference on Visualization* (2010).
[6] Hamish Carr, Jack Snoeyink, and Ulrike Axen. 2000. Computing contour trees in all dimensions. *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms* (2000).
[7] J. Eldridge, M. Belkin, and Y. Wang. 2014. Denali: A tool for visualizing scalar functions as landscape metaphors. *OSU Technique Report, Software: http://denali.cse.ohio-state.edu/* (2014).
[8] Manuel Freire, Catherine Plaisant, Ben Shneiderman, and Jen Golbeck. 2010. ManyNets: An Interface for Multiple Network Analysis and Visualization. *SIGCHI* (2010), 213–222.
[9] Thomas M. J. Fruchterman and Edward M. Reingold. 1991. Graph Drawing by Force-directed Placement. *Softw. Pract. Exper.* (1991).
[10] Martin Gronemann and Michael Junger. 2012. Drawing Clustered Graphs as Topographic Maps. *Graph Drawing* (2012).
[11] Frank Ham, Hans-Jorg Schulz, and Joan M. Dimicco. 2009. Honeycomb: Visual Analysis of Large Scale Social Networks. *Proceedings of the 12th IFIP TC 13 International Conference on Human-Computer Interaction: Part II* (2009).
[12] W. Harvey and Y. Wang. 2010. Generating and Exploring a Collection of Topological Landscapes for Visualization of Scalar-Valued Functions. *Eurographics / IEEE-VGTC Conference on Visualization* (2010).
[13] Keith Henderson, Brian Gallagher, Tina Eliassi-Rad, Hanghang Tong, Sugato Basu, Leman Akoglu, Danai Koutra, Christos Faloutsos, and Lei Li. 2012. Structural Role Extraction and Mining in Large Graphs. *ACM SIGKDD* (2012).
[14] Xin Huang, Hong Cheng, Lu Qin, Wentao Tian, and Jeffrey Xu Yu. 2014. Querying k-truss community in large and dynamic graphs. *SIGMOD* (2014).
[15] David Koop, Juliana Freire, and Claudio T. Silva. 2013. Visual summaries for graph collections. *PacificVis* (2013).
[16] Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford Large Network Dataset Collection. http://snap.stanford.edu/data. (2014).
[17] Shawn Martin, W. Michael Brown, Richard Klavans, and Kevin W. Boyack;. 2011. OpenOrd: an open-source toolbox for large graph layout. *Proc. SPIE* 7868 (2011), 786806–786806–11.
[18] Yiye Ruan and Srinivasan Parthasarathy. 2014. Simultaneous Detection of Communities and Roles from Large Networks. *ACM COSN* (2014).
[19] Ahmet Erdem Sariyuce, C. Seshadhri, Ali Pinar, and Umit V. Catalyurek. 2015. Finding the Hierarchy of Dense Subgraphs using Nucleus Decompositions. *WWW* (2015).
[20] Natascha Sauber, Holger Theisel, and Hans-Peter Seidel. 2006. Multifield-Graphs: An Approach to Visualizing Correlations in Multifield Scalar Data. *IEEE TVCG* (2006), 917–924.
[21] Lei Shi, Qi Liao, Hanghang Tong, Yifan Hu, Yue Zhao, and Chuang Lin. 2014. Hierarchical Focus+Context Heterogeneous Network Visualization. *PacificVis* (2014).
[22] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. LINE: Large-scale Information Network Embedding. *WWW* (2015).
[23] A. Telea and O. Ersoy. 2010. Image-based edge bundles: simplified visualization of large graphs. *Eurographics / IEEE-VGTC conference on Visualization* (2010).
[24] Robert van Liere and Wim de Leeuw. 2003. GraphSplatting: Visualizing Graphs As Continuous Fields. *IEEE TVCG* (2003).
[25] T. von Landesberger, A. Kuijper, T. Schreck, J. Kohlhammer, J.J. van Wijk, J.-D. Fekete, and D.W. Fellner. 2011. Visual Analysis of Large Graphs: State-of-the-Art and Future Research Challenges. *Computer Graphics Forum* (2011).
[26] Jia Wang and James Cheng. 2012. Truss Decomposition in Massive Networks. *Proc. VLDB Endow.* (2012), 812–823.
[27] Nan Wang, Srinivasan Parthasarathy, Kian-Lee Tan, and Anthony K. H. Tung. 2008. CSV: Visualizing and Mining Cohesive Subgraphs. *ACM SIGMOD* (2008).
[28] Nan Wang, Jingbo Zhang, Kian-Lee Tan, and Anthony K. H. Tung. 2010. On Triangulation-based Dense Neighborhood Graph Discovery. *Proc. VLDB Endow.* (2010), 58–68.
[29] Gunther Weber, Peer-Timo Bremer, and Valerio Pascucci. 2007. Topological Landscapes: A Terrain Metaphor for Scientific Data. *IEEE TVCG* (2007).
[30] Rephael Wenger. 2013. *Isosurfaces: Geometry, Topology, and Algorithms.* CRC Press.
[31] Zhiqiang Xu, Yiping Ke, Yi Wang, Hong Cheng, and James Cheng. 2012. A Model-based Approach to Attributed Graph Clustering. *ACM SIGMOD* (2012).
[32] Jaewon Yang and Jure Leskovec. 2013. Overlapping Community Detection at Scale: A Nonnegative Matrix Factorization Approach. *ACM WSDM* (2013).
[33] Xintian Yang, Sitaram Asur, Srinivasan Parthasarathy, and Sameep Mehta. A visual-analytic toolkit for dynamic interaction graphs. In *SIGKDD 2008.*
[34] Yang Zhang and Srinivasan Parthasarathy. 2012. Extracting Analyzing and Visualizing Triangle K-Core Motifs within Networks. *IEEE ICDE* (2012).
[35] Yang Zhang, Yusu Wang, and Srinivasan Parthasarathy. 2017. Analyzing and Visualizing Scalar Fields on Graphs. *arXiv:1702.03825 [cs.DB]* (2017).
[36] Yang Zhou, Hong Cheng, and Jeffrey Xu Yu. 2009. Graph Clustering Based on Structural/Attribute Similarities. *VLDB* (2009).