Graph Visualization Through Collaborative Node Grouping

ABSTRACT

The node-link graph visualization with more than a hundred nodes is a common yet challenging task. Traditional methods either create the layout with an abundance of visual clutters and slower the analysis in details, or oversimplify the topology through graph clustering and filtering. In this paper, we propose a novel method, namely Collaborative Node Grouping (CNG). CNG condenses the graph and reduces the visual complexity without losing any graph information in the resulting representation. The CNG-compressed graph preserves many critical features from the original graph, making it easy for human understanding and analysis. CNG supports directed, weighted and dynamic graphs through natural extensions. We have conducted case and user studies in several real-world scenarios, e.g. over the security and traffic graphs. Results show that our method is effective both quantitatively in terms of the graph compression rate and qualitatively from the user's feedbacks. Performance evaluation also confirms that CNG can scale to graphs with a million nodes in a standard desktop machine.

1 INTRODUCTION

Graphs representing the relationship data are one of the most popular information genres in reality. Real-world graphs are essentially large, especially lately as the human's ability to retrieve and aggregate information grows tremendously. Understanding these large graphs are crucial in many cases. For example, a network/cloud administrator needs to keep track of the traffic distribution among servers and hosts for a better network/virtual-machine optimization during the resource planning stage. Upon security events, they also need to access the latest traffic graph to increase situation awareness for more responsive troubleshootings. In a broader Internet scenario, an SNS website owner/user depends on the knowledge of the global/egocentric social network to design more effective promotion strategies to expand the network or for online advertising.

Visualization is a well-known technology among a short list to access and analyze large graphs. However, visualizing a graph with more than a hundred nodes faces two fundamental challenges. First, the classical force-directed methods in most cases fail to calculate an optimally aesthetic graph layout in real time ($\sim$1s). Second, even if a huge graph layout is computed, the visual clutters (mainly the edge crossings) created by the straightline node-link representation prohibit the user from understanding the graph in details, which is important for analytical tasks.

Existing solutions to overcome these challenges fall into three categories. The first class focuses on the efficient drawing algorithms of the huge graph [22] [26] [17], balancing the computation time and layout aesthetics. The algorithms reduce the visual clutters by minimizing the edge crossings, but still the non-planar real-world graphs are too dense to understand by the users and hard for the drill-down analysis on the details. The second class solves the problem by directly reducing the complexity of the underlying data, through graph clustering [5] [4] [27] and filtering [24] [33]. This approach can create abstracted graphs simple enough for human understanding and feasible as the starting point of an analysis. The drawbacks are also clear, the abstracted graph often distorts much from the original topology and many critical details and contexts are lost which can be misleading in the analysis process. The third class are the best-effort methods to alleviate the visual clutter without reducing the data. They include the edge bundling approaches to visually cluster group of edges, and the huge graph navigation methods through hyperbolic views [28] [29] and the fisheye distortion [15]. Still there are other drawing formats of the huge graph rather than the node-link representation, e.g. the matrix [19] and NodeTrix [20], but these will be out of the scope of this paper.

In this paper, our ultimate goal is to create an abstracted and smaller-sized node-link representation of the huge graph without losing its topology and detailed graph information, so that both the layout computation complexity and the visual complexity are reduced simultaneously. We are inspired by the overwhelming broadcast patterns in the security graphs (Figure 7(a)), where the numerous standalone recipients are in the same position within the graph. This creates considerable topology redundancies as well as the unnecessary visual clutters. Our idea is to condense the graph by removing the topology redundancy while keeping the whole graph information intact, rather than the clustering and filtering which can drop valuable pieces.

In more detail, we propose the Collaborative Node Grouping (CNG) graph compression method. CNG groups the nodes with the same neighbor set together as a larger mega-node and regenerates a compressed graph for the subsequent visualization and analysis. A similar pairing idea has been introduced by Davis and Hu [10] in the huge graph coarsening process, but only pairwise grouping is considered. Also the visualization in [10] still works on the original graph. Beyond the CNG of the basic graph, we extend in this paper to support the directed, weighted and dynamic graphs evolving over time, which are more useful in reality and for many task-specific graph analytics. To further reduce the graph complexity, we also introduce the fuzzy CNG method according to the neighbor set based pairwise node similarity. It is shown through evaluations over the real-world graphs that, in most cases, the visual complexity (measured by the number of edges) can be reduced more than ten times after a basic CNG. The computational complexity introduced by CNG is much lower than the latest applicable huge graph layout algorithm [22], scaling to support graphs with a million nodes.

Towards a user-friendly representation of the CNG-compressed graph, we have customized the visual encodings of the compressed graph visualization. Many interactions are introduced to accelerate the compressed graph visual analysis, including the graph navigation methods, the switches between the compressed and original graphs, and the user-defined node groupings through simple mouse actions. Stable layouts are computed in a best-effort manner to keep the user's momentum between consecutive graph views, along with the staged animations to smooth the visual transition.

Finally, we demonstrate through four case studies that graph visualization tools with CNG can significantly improve the domain user's capability in their graph analysis related tasks. For example, in a situation awareness scenario, the security admin can discover the potential noteworthy events with the CNG-enabled traffic analysis tool more easily and quickly. Controlled studies and user feedbacks also show better performance and adoption of the CNG-enabled graph visualization tool in the large graph understanding and detail-accessing aspects, compared to the traditional graph views employed by the commercial tools.

2 Related Work

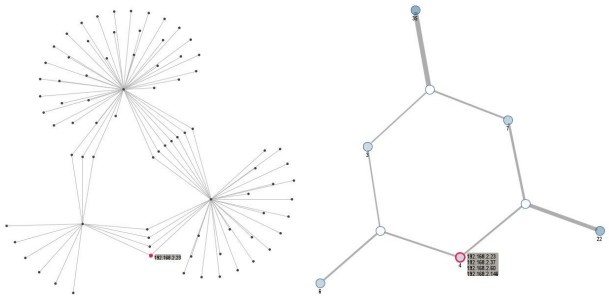
Classical force-directed algorithms with either the spring embedder [12] [13] or the graph theoretic distance based stress model [25] [16] generate visually aesthetic layouts for small graphs, but fail to scale to graphs with a thousand nodes, even with certain aesthetics trade-offs. The root cause is the over $O(N^2LogN)$ computational complexity. Therefore, the major theme of the huge graph drawing algorithms is the balance of the computational complexity and the layout quality. In this part, we shall focus on the algorithms for the straight-line drawing of huge general graphs.

A number of huge graph drawing algorithms [14] [22] apply an iterative coarsening process to reduce the graph to smaller ones until the resulting graph can be handled by the classical algorithms. Then the graph layouts are refined recursively from the finest coarsened graph back to the original huge graph. This is generally called the multi-level approach. These algorithms differ in the coarsening, refining and the methods to layout the graph recursively. The coarsening can be through the pairwise node grouping based on the independent edge collapsing [18], or directly replacing the graph with the independent vertex set [6], or a filtered vertex set [14]. The refinement of each level of graph can be through the classical layouts [22] or computed by the weighted center of several landmark nodes in the previous level [11]. The layout algorithms are generally the classical ones, e.g. the spring embedder [22], stress-model [14] and MDS [11]. The multi-level approaches scale to general graphs of 10^5 nodes with a reasonable time and layout quality.

For graphs in an extreme scale (e.g. with millions of nodes), the high-dimensional embedding (HDE) algorithm [17] is proposed. HDE chooses k centers from the huge graph and forms a k-dimensional coordinate system by the distance to all the k centers. Each node with the k-dimensional coordinate is projected back to the two dimensions through carefully selected linear combinations. ACE [26] is another fast algorithm to draw extreme-scale graphs. ACE is based on the Hall's model to minimize a pairwise weighted quadratic energy. Both ACE and HDE can compute a drawing for graphs with $10^5 \sim 10^6$ nodes in a minute, but its usage are limited to grid-like graphs. For the real-world huge graphs with scale-free and small-world features, their layouts are not satisfying. For a more extensive introduction of huge graph drawing algorithms, the reader may refer to the survey by Hu [23].

In parallel with the direct drawing, another category of methods visualize the huge graph through the data reduction. A number of works in the literature generate hierarchies from the graph by geometric clustering [30], graph-theoretic clustering, multi-level graph partitioning [5] or graph coarsening [15]. The resultant hierarchical tree helps to create a multi-level view [30] [15] or a clustered view [5] [4] [27] of the huge graph. These visual abstractions work well in showing the graph topology for an overview purpose by hiding many details. To access the details, some provides navigation methods to traverse the graph hierarchy [4]. The second graph reduction method tries to filter the weak edges according to the edge betweenness centrality [24]. A minimal spanning tree can be constructed to significantly reduce the visual clutter and leave a backbone of the graph [33]. Graph nodes can also be filtered by the ranking of any node/link attribute values. The PivotGraph [34], a similar work to ours, groups nodes by their attribute values, but does not focus on showing the graph topology.

The other works on the huge graph visualization can be orthogonal to the above methods. The edge bundling approaches [21] [9] carefully select control points for each edge according to the graph structure. The edges are then geometrically clustered through the control points to reduce the visual clutter. The hyperbolic visualizers [28] [29] layout the tree-like large graph in a space-infinite hyperbolic plane and then map the graph back to the Euclidean space. The viewing and navigation experience is rather pleasant when applying to the large graphs. The other navigation methods, e.g. the "overview + detail" through a tree navigator [4] and the Degree-of-Interest based context subgraph [32] are also applicable in the huge graph visualizations.



(a) Original graph (b) Compressed graph

Figure 1: Collaborative node grouping on a security graph.

3 Collaborative Node Grouping and The Compressed Graph

3.1 Basic Idea

The basic idea of our approach is to aggregate nodes with the same neighbor set in the graph together into groups and construct a new graph for visualization. Our idea is motivated by the observation that most real-world graphs have considerable topology redundancy. For example in Figure 1(a), the host "192.168.2.23" connects to two hub nodes and is surrounded by three other hosts with exactly the same connection pattern. Though visually aesthetic, the redundant nodes and links distract the user in understanding the graph information, especially when such a graph is embedded as part of a large graph.

We name our approach Collaborative Node Grouping (CNG), after the Collaborative Filtering (CF) technique developed in the web and recommendation researches. CF predicts the interests of a user from the other users with the same preference (buying the same set of items). Essentially, CNG over the user-item bipartite graph will group the users according to their item preferences, and items according to the user group.

The new graph after CNG is called the *compressed graph* (Figure 1(b)) from the *original graph* (Figure 1(a)). Compared to the existing large graph layout, clustering and filtering approaches, the compressed graph achieves both significant visual complexity reduction (Section 4.5) and zero graph information loss in the representation (Section 3.2). Since the compressed graph is still in a node-link format, the existing approaches, especially the graph navigation methods, can be applied over CNG to further improve the visualization.

3.2 The Compressed Graph

The compressed graph has two kinds of nodes (Figure 1(b)): the *single-node* from the original graph (hollow) and the *mega-node* grouped from multiple *sub-nodes* in the original graph (filled). Each link in the compressed graph is aggregated from the links between every pair of sub-nodes/single-nodes belonging to the two endpoints respectively. One basic property of the compressed graph is, each mega-node will either have no intra-group link or the mega-node is a clique. Note that in the former case, the sub-nodes within each mega-node are connected by 2-hop in the original graph. The compressed graph preserves many valuable features from the original graph (except for the fuzzy CNG), making it eligible for the visual representation.

Graph integrity. We validate this property by showing that the original graph can be restored uniquely from the compressed graph: first, collapse each mega-node into a collection of isolated nodes or a clique of nodes according to the type and size encoding associated

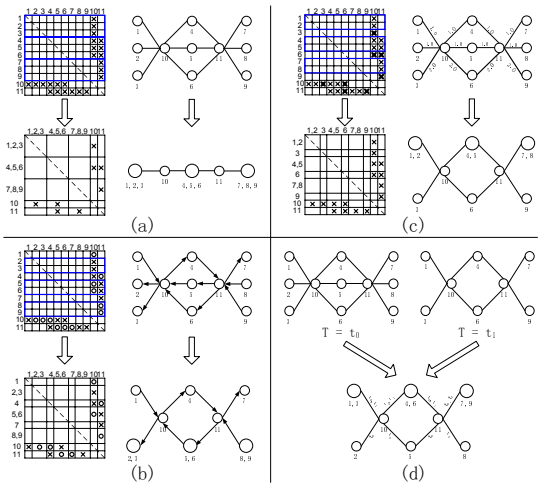


Figure 2: CNG algorithm examples, from the original graph to the compressed graph. The left part of each subfigure is the graph adjacency matrix and the right part is the corresponding node-link graph. The smaller nodes indicate the single-nodes and the larger nodes in the compressed graph indicate the mega-nodes. (a) Basic CNG; (b) CNG for directed graphs. "x" in each matrix cell indicates a "to" direction, "o" indicates a "from" direction; (c) CNG for weighted graphs. The weight is labeled on each link; (d) CNG on a dynamic graph with two time windows.

with the mega-node; second, add a link between every two nodes whose previous mega-nodes have a link in the compressed graph.

Shortest path. Consider any two nodes A and B in the original graph and assume A and B are not grouped together in the compressed graph. The shortest path from the mega-node of A to the mega-node of B in the compressed graph is the path by replacing each node in the original "A to B" shortest path to their mega-nodes in the compressed graph, with the same path length. For nodes within the same mega-node, their shortest path is of length one or two.

The proof is completed by showing that any two nodes in such a shortest path of the original graph can not be grouped in the compressed graph. Consider node C and D on the shortest path from A to B, in case C and D can be grouped together, indicating the same connection pattern of C and D, then C can directly link to the next hop of D and also the preceding hop of C can directly link to D, making the path not a shortest one.

Graph connectivity. By the shortest-path preserving property, there will be no shortest path between two mega-nodes in the compressed graph unless there is one between the original nodes. Therefore, the disconnected nodes are still disconnected in the compressed graph. This property also suggests that CNG can be conducted separately over each connected component of a large graph.

Visual node affinity. Since most layout algorithms employ shortest-path distance based heuristics, the nodes with a smaller shortest path in the original graph and are closer visually will also be closer to each other in the compressed graph as mega-nodes.

Uniqueness. Each original graph will have only one compressed graph after the basic CNG, since the algorithm is deterministic (Section 4.1).

4 Compression Algorithm

In this section, we introduce the basic CNG compression algorithm and its extensions to support several graph variations and the control of the compression rate. Before that, we first define the terminologies used throughout the algorithm description.

Definition. Let $G = (V,E)$ be a directed, weighted and connected original graph where $V = \{v_1,...,v_n\}$ and $E = \{e_1,...,e_m\}$ denote the node and link set. Let W be the graph adjacency matrix where $w_{ij} > 0$ indicates a link from v_i to v_j, with w_{ij} denoting the link weight. In each row of W, $R_i = \{w_{i1},...,w_{in}\}$ denotes the row vector for node v_i, representing its connection pattern. The compressed graph after CNG is denoted as $G^* = (V^*,E^*)$. The compression rate is defined by $\Gamma = 1 - \frac{|V^*|}{|V|}$ ($1 - \frac{|E^*|}{|E|}$ in Section 4.5).

4.1 Basic Algorithm

The basic algorithm takes the graph as a simple, undirected and unweighted one by setting $w_{ii} = 0$ and $w_{ij} = w_{ji} = 1$ for any $w_{ij} > 0$.

Basic Collaborative Node Grouping. On graph G, order its node list by the corresponding row vectors $R_i (i = 1,...,n)$. For any collection of nodes with the same row vector (including the single outstanding node), aggregate them into a new mega-node/single-node $Gv_i = \{v_{i_1},...,v_{i_k}\}$. All Gv_i form the node set V^* for the compressed graph G^*. Also let $fv_i = v_{i_1}$ denote the first sub-node in Gv_i. The link set E^* in G^* are generated by simply replacing all fv_i with Gv_i in the original link set, and removing all the links not incident to any fv_i.

Figure 2(a) gives an example of the basic CNG over a small graph with 11 nodes.

Completeness: Any two nodes in the compressed graph G^* have different row vectors.

4.2 Extensions

Real-world graphs are often directed, weighted and evolving over time, we extend the basic CNG algorithm to support these natures by generalizing the definition of the adjacency matrix and the corresponding row rectors.

Directed Graph. The adjacency matrix W is unfolded to encode the connection directions for each node. Each row vector $R_i (i = 1,...,n)$ becomes $R_i = \{w_{i(-n)},...,w_{i(-1)},w_{i1},...,w_{in}\}$ having $w_{i(-j)} = w_{ji}$ for $j = 1,...,n$. Figure 2(b) gives an example.

Weighted Graph. The adjacency matrix W is switched to the weighted one by mapping one numeric data attribute of link (i,j) (e.g. flow count in a traffic graph) to w_{ij} in the matrix, as shown in Figure 2(c). To further increase the compression rate, discretization of the link weight is allowed: first transform all link weights into $w_{ij} \in (0,1]$ by either linear or non-linear normalization, then pick a bin count $B (B \geq 1)$ and regenerate link weights by $w_{ij} = \lceil w_{ij} \times B \rceil$.

Dynamic Graph. The dynamic graph is constructed by aggregating graphs in consecutive time windows and recording the timestamp on each link upon the aggregation. Given a dynamic graph $G_T (T = [T_0,T_1])$, on each link (i,j) of G_T, there will be a timed list $L_{ij} = (t_0 : w_{ij}^0,...,t_c : w_{ij}^c), T_0 \leq t_0 < ... < t_c \leq T_1$, indicating the link's temporal pattern with $w_{ij} = \sum_{k=1}^{c} w_{ij}^k$. To apply CNG on dynamic graphs, the adjacency matrix W and the row vector R_i are updated by replacing the numeric w_{ij} to the list-valued L_{ij}. An example is given in Figure 2(d). Similarly to the weighted CNG, the time window size can be tuned to adjust the compression rate. Given a time window count B, each timestamp t_k in L_{ij} is discretized by $t_k = \lceil \frac{t_k - T_0}{T_1 - T_0} \times B \rceil$.

The rest of the algorithm for these graph variations follows the basic CNG algorithm. Note that our treatments on the directed, weighted and dynamic graphs are orthogonal to each other, therefore can be combined together to deal with graphs with two or three of these natures. The visual explanation of the CNG extensions are also shown in Figure 2: beyond the basic CNG, the extensions not only consider the neighbor set of each node, but also consider the different kind of connections between the node and each of its neighbors, including the direction ("to", "from", "bidirectional"), frequency/weight and the temporal pattern.

Supporting Clique.

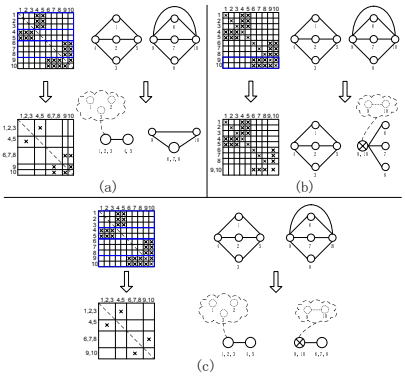


Figure 3: Impacts of the diagonal cells to CNG results on undirected and unweighted graphs. (a) Set $w_{ii} = 0$, the mega-nodes have no intra-group links; (b) Set $w_{ii} = 1$, the mega-node as a clique is supported, but most other nodes are prohibited in the CNG; (c) The optimal CNG where the two approaches are combined.

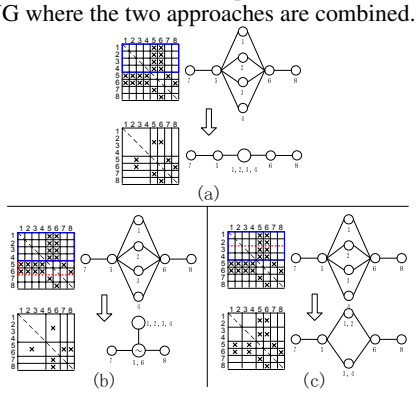


Figure 4: CNG with different compression rates: (a) Original CNG; (b) Fuzzy CNG with similarity threshold $\xi = 0.66$; (c) CNG compression level control with $\beta = 0.66$.

As shown in Figure 3(a), one feature of the basic CNG is that the nodes grouped together do not have any intra-group link. This actually ensures zero information loss in the compressed graph. However in some other cases, as shown in Figure 3(b), it is also useful to group a clique of nodes with the same external connection together. Specific rendering as in Figure 3(b) can be applied to differentiate between compressed nodes with isolated sub-nodes and fully connected sub-nodes (clique).

A straightforward method is to set $w_{ii} = 1$ for all the diagonal cells. But this prohibits the grouping of isolated nodes (Figure 3(b)). To achieve the optimal CNG performance as in Figure 3(c), we devise a two-step hybrid approach. In the first step, the graph adjacency matrix W is set to $w_{ii} = 0$, allowing the grouping of isolated nodes. In the second step, W is reset to $w_{ii} = 1$ and all the original nodes not aggregated in the first step are grouped again.

4.3 Controlling the Compression Rate

As introduced above, CNG is a deterministic algorithm that for the same original graph, it always produces the same compressed graph and also the same visualization. In the real usage, the user would like to control the level of details after the compression: on one hand, for graphs still large after CNG, more node groupings are expected to further reduce the visual complexity for analysis; on the other hand, for graphs with a high compression rate, the user may want to roll back a little to compensate for the topology mental map.

Fuzzy Collaborative Node Grouping. The basic idea of fuzzy CNG is to group nodes with not only the same, but also the similar neighbor set, as illustrated in Figure 4(b). Then the compression rate can be increased by compensating with some information deviation. The key is to define the pairwise similarity score between graph nodes under CNG. Here we adopt the standard Jaccard similarity coefficient between two sample sets A and B by $J(A,B) = \frac{|A \bigcap B|}{|A \bigcup B|}$. To also extend the definition to the directed and weighted graph, we introduce a unified Jaccard similarity computation between node v_i and v_j in graph G by $\rho = \frac{\sum_{\forall k} \min(w_{ik}, w_{jk})}{\sum_{\forall k} \max(w_{ik}, w_{jk})}$. Note that for the undirected graph, $k = 1, ..., n$, and for the directed graph, $k = -n, ..., -1, 1, ..., n$. Fuzzy CNG is achieved by setting a similarity threshold ξ, the pair of nodes with $\rho \geq \xi$ are grouped together iteratively.

Level of Detail. In contrast to the fuzzy CNG, Level Of Detail (LOD) control allows the user to access to more details beyond the fully compressed graph, with a lower compression rate. As shown in Figure 4(c), the major gain is to maintain the user's mental map to a certain kind of graph topology, while CNG or fuzzy CNG will potentially affect the mental map by grouping too many nodes together.

LOD is achieved in CNG framework by re-splitting the aggregated mega-node into smaller mega-nodes of the same size. By default without LOD, the CNG compression level is set to 100%. Using a compression level of β, each mega-node containing s sub-nodes is partitioned into k smaller mega-nodes where $k = \lfloor 1 + (s-1) \times (1-\beta) \rfloor$.

4.4 Implementation

Collaborative Node Grouping. The key to implement CNG is to group nodes with the same row vector. It can be achieved through an appropriate hash function $H(R_i)$ over the row rector identifiers, and a hash collision resolution mechanism (e.g. HashMap in Java implementation). A basic method to create the row vector identifier is to splice the positive cells into a string, attached with the destination node IDs. For example, the identifier for R_4 in Figure 2(a) can be "10:1,11:1", where ":" and "," are used as delimiters. Directed, weighted and dynamic graphs can also be supported by encoding the extended row vectors to identifiers accordingly.

The hash-based implementation has a computation complexity of $O(ND) = O(E)$, where N is the number of nodes in the original graph for the traversal, D is the average node degree for splicing the row vector identifier, E denotes the number of links in the original graph. For most graphs, D is much smaller than N and can be considered a bounded constant. Therefore, the hash-based implementation of the deterministic CNG works well even for very large graphs, as shown in Table 1∼4.

Fuzzy Collaborative Node Grouping. To implement fuzzy CNG, a greedy approach is applied. For each node in the original graph, the similarity score between this node and all the nodes already traversed are computed until one score falls above a threshold $(1+\xi)/2$. Then the two nodes are grouped together. Note that we introduce an approximation here to reduce the complexity: after each node grouping, the new mega-node is represented by the first sub-node in the group, called the anchor node; for the following similarity computation with this mega-node, only the row vector of the anchor node is used. Due to the properties of the Jaccard coefficient, it is guaranteed that for any two sub-nodes in the same mega-node after this fuzzy CNG implementation, their similarity score will be above ξ (we omit the proof here). The computation complexity for this fuzzy CNG implementation is $O(NN^*D) = O(N^2D(1-\Gamma))$ where N^* is the number of nodes in the compressed graph due to the pairwise similarity computation. For large graphs where CNG does not have a significant compression rate, the performance will be considerable low, as shown in the line 3 of Table 4.

Shingle Ordering of Row Vectors. We introduce the shingle ordering of row vectors as a fast algorithm for the fuzzy CNG on

Table 1: CNG performance evaluation on VAST Challenge dataset.

Data	nodes (before)	edges (before)	nodes (after)	edges (after)	rate (edges)	time (compress)	layout (before)	layout (after)
undirected sim=1	409	1613	17	50	96.9%	0.007	0.24	0.084
undirected, sim=0.8	409	1613	16	39	97.6%	0.012	0.242	0.088
undirected, sim=0.5	409	1613	13	23	98.6%	0.006	0.25	0.079
directed sim=1	409	1613	26	82	94.9%	0.005	0.245	0.084

Table 2: CNG performance evaluation on Honeypot dataset.

Data	nodes (before)	edges (before)	nodes (after)	edges (after)	rate (edges)	time (compress)	layout (before)	layout (after)
undirected	15380	16353	2	2	99.9%	0.123	10.179	0.079
undirected weighted #bin=10	15380	16353	5	8	99.9%	0.151	10.179	0.692
undirected	44668	45582	2	2	99.9%	0.401	35.09	0.08
undirected	1051595	1158150	2	2	99.9%	4.56	(500) 36.404	0.026
undirected dynamic #win=1	43602	47752	9	16	99.9%	1.27	33.504	0.1
undirected dynamic weighted #win=1 #bin=10	43602	47752	105	208	99.6%	1.102	33.504	0.946

the unweighted graph. For each row vector R_i, construct the element set $A_i = \{a | w_{ia} = 1\}$. Given any permutation $\sigma : \{1,...,n\} \rightarrow \{1,...,n\}$, the shingle of the row vector R_i is defined by $M_\sigma(A_i) = \sigma^{-1}(min_{\alpha \in A_i}\{\sigma(\alpha)\})$. By shingle properties [8], the probability shingles of set A and B are identical is precisely their Jaccard coefficient $J(A,B)$.

Picking several independent permutations (e.g. min-wise independent family [7]) and computing the frequency of having the same output shingles will approximate the similarities among all row vectors. By using elaborately designed data structure, the computation complexity can be reduced to $O(kN^2/N^*) = O(kN/(1-\Gamma))$ where k is the number of permutations.

4.5 Performance

We evaluate the CNG performance in terms of the visual compression rate (by the number of edges), the compression time and the layout time before and after. While the running time varies on different systems, all the evaluations are carried out on the same 64-bit Windows desktop (Intel Core i7@3.40GHz with 8GB memory). Table 1∼Table 4 show the performance results. Notably for most data sets except for the social graphs, CNG achieves a more than 90% compression rate with the basic algorithm or applying a fuzzy setting. The deterministic CNG can scale to a million of nodes and multi-millions of edges with a reasonable computation time. The fuzzy CNG with the optimized shingle implementation supports up to graphs with 10^5 nodes and returns results in half a minute.

5 Compressed Graph Visualization

We introduce the visual mapping of the CNG-compressed graph in this section, along with the interactions to accelerate the graph analysis.

5.1 Visual Encoding

The right panel of Figure 5 gives an example of the compressed graph visualization after the basic CNG. As shown in the figure, the mega-nodes are differentiated from the single-nodes by the node fill color. The single-nodes have no fill and the mega-nodes have standard fill colors, with the color saturation mapped to the number of sub-nodes within the group. The larger group is filled with the more saturated color. By default, the fill color hue is blue, e.g. the top-right node "192.168.1.10+". For the mega-nodes created by the fuzzy CNG, e.g. the one with a label "192.168.2.11+", the node fill color will gradually shift to green and then to brown, according to the smallest pairwise similarity score within the group. The mega-node containing sufficiently dissimilar nodes will lead to a pure brown fill color. We do not use the node size to represent the group size of the mega-node, since the group size normally has a rather biased distribution. The large groups will introduce unnecessary visual complexities which counters our initial design goal.

Node labels of the mega-node are created by aggregating the labels of the sub-nodes in the original graph. Due to the space limitation, an abstracted label is drawn on each mega-node as the node identifier. The full label is only visible upon a mouse-over or click

Table 3: CNG performance evaluation on Data Center dataset.

Data	nodes (before)	edges (before)	nodes (after)	edges (after)	rate (edges)	time (compress)	layout (before)	layout (after)
undirected	6509	18347	433	2626	85.7%	0.538	4.885	0.402
undirected weight #bin=10	6509	18347	434	2636	85.6%	0.132	4.885	0.37
undirected compress=0.8	6509	18347	1565	5428	70.4%	0.151	4.885	0.427
undirected sim=0.5	6509	18347	340	1510	91.8%	0.364	4.885	0.336

Table 4: CNG performance evaluation on Social Network dataset.

Data	nodes (before)	edges (before)	nodes (after)	edges (after)	rate (edges)	time (compress)	layout (before)	layout (after)
tweet undirected	88382	122976	27715	66858	45.6%	2.074	too long	(500) 0.522
tweet undirected sim=0.5 shingle	88382	122976	25292	59873	51.3%	24.92	too long	(500) 0.547
tweet undirected sim=0.5 anchor	88382	122976	26672	63089	48.7%	370.121	too long	(500) 0.733
comment undirected	229566	320786	76415	200034	37.6%	3.653	too long	(500) 0.502
retweet undirected	306139	1294210	159639	1088208	15.9%	20.797	too long	(500) 139.806
follower undirected	263175	2926986	182835	2574823	12%	32.835	too long	(500) 95.1

action, as on the node "192.168.1.10+". The group size of each mega-node is drawn below the visual node, together with the intra-group similarity score if applicable. By default, straight lines are used to represent the links in the compressed graph, with the line thickness mapped to the summed value of the link counts of all the corresponding links in the original graph.

In the implementation, we also support flexible mappings of the compressed graph attributes into the visuals. For example, the mega-node fill color can be set by the summed node degree from the original graph. The node label can be any information attached, e.g. the alphabetical icons on each node of Figure 5 indicate the type of anomalies happen on the node. The link thickness can also show the number of original links, or the maximal/minimal/average/median attribute value within the aggregated link group, other than the summed value.

For the dynamic graph CNG, the link color is used to indicate the different temporal patterns on the link. As shown in Figure 11(c), each link is drawn as several colored segments from the selected node. Each segment corresponds to one time window on the data, with the color ranging from orange to blue. The length of each segment is proportional to the summed link weight on the time window.

5.2 Interaction Design

The visualization supports several basic graph interactions, including the geometric zoom&pan, node drag&drop, node/link highlight&selection, as well as the various node/link visual mappings mentioned above. Beyond that, more controls over the CNG setting are accessible through a control panel as in the left of Figure 5. In the "Compression Options" section, multiple checkboxes work as switches for the basic, directed, weighted and dynamic graph CNG. For the weighted graph CNG, the link weight mapping from the graph attribute can be specified. In case the link weight is normalized, a bin number can be selected to discretize the link weight. For the dynamic graph CNG, time window size can be set to allow different temporal granularity in the graph aggregation. In the "Compression Level" and "Compression Similarity" sections, a larger or smaller compressed graph is tunable by the LoD control and the fuzzy CNG over the basic algorithm, as described in Section 4.3.

Complementary to the automatic CNG compression, we also introduce the manual node grouping/splitting interaction as in many cases the users have their own criteria towards a best graph abstraction. This interaction is applicable for both the original graph and the compressed graph with any CNG settings. In a manual grouping process, the user can either select a collection of nodes and click the "group" button in the navigation panel, or use drag-and-drop to group one node into another once per time. In the drag-and-drop process, the pairwise similarity scores with all the other nodes are shown as visual hints to facilitate the interaction. The resulting node has the same visual mapping with the mega-node in the compressed graph. In a manual splitting process, the user can either select some mega-nodes and click the "split" button, or just double-click one mega-node. The mega-node grouped by the fuzzy

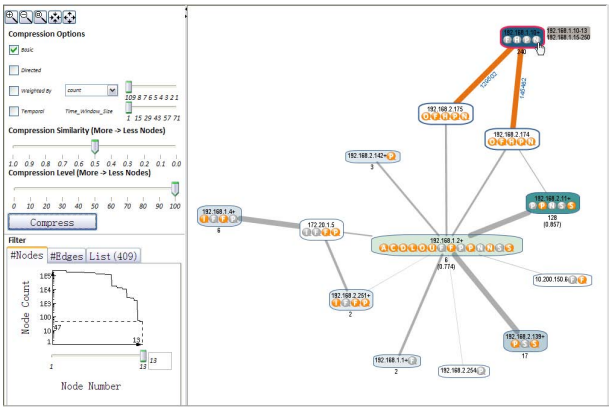


Figure 5: The user interface for the compressed graph visualization: the left panel includes various controllers for the compression operation, the right panel is the main window for the graph visualization. The graph data is the security traffic flows in VAST Challenge 2011 data set.

CNG (intra-group similarity score below one) will collapse to several mega-nodes after the basic CNG, and the mega-node after the basic CNG will collapse to the original sub-nodes. Further, if a double-click happens on a single node, this node will be checked to group with all the other single/mega-node in the graph under the current CNG setting.

Both the compressed graph and the original graph support some existing large graph analysis interactions. A useful tool is the node/edge filtering as in the bottom of the control panel in Figure 5. In the "node" tab, the user can filter the graph according to the node importance. By default, the node count by summing its incident edge counts is used. A node count distribution is shown on top of the slider to suggest a better filter setting. In the "edge" tab, the graph is filtered according to the edge count, the nodes not incident to any important edge are removed in the final graph. The mapping of the node/edge filtering criteria can be manually adjusted according to the available graph data attributes.

5.3 Layout and Animated Transition

We apply the force-directed layout algorithm in Kamada-Kawai model [25] with the stress majorization solver [16] for most compressed graphs with less than 10^3 nodes. For the original huge graph and the compressed graph still large, the multi-level force-directed layout algorithm from GraphViz tool [22] is used, which scales to graphs with 10^5 nodes in a reasonable time.

However, the key challenge here is not on the layout computation, but on how to keep the user's visual momentum across consecutive views, especially when switching between the original graph and the compressed graph. In this paper, we introduce two orthogonal methods to achieve that: a stable layout method to minimize the gap between the consecutive graph views, and a customized animated transition "morphing" from the previous layout to the current.

The stable layout method is integrated with the iterative solver of the Kamada-Kawai model. The goal is to set an initial layout for the current graph as close as possible to the previous one, then the layout after computation will be reasonably stable without losing the graph aesthetics. The core idea is to maintain the latest positions for both the mega-nodes in the compressed graph and all the single/sub-nodes in the original graph. When manipulating the compressed graph, the latest positions of each sub-node is also updated along with the mega-node it belongs to. The initial layout of a mega-node will be its latest position if it is in the previous graph. Otherwise, the latest positions of all its sub-nodes are retrieved and the average position is taken as the initial layout of the mega-node.

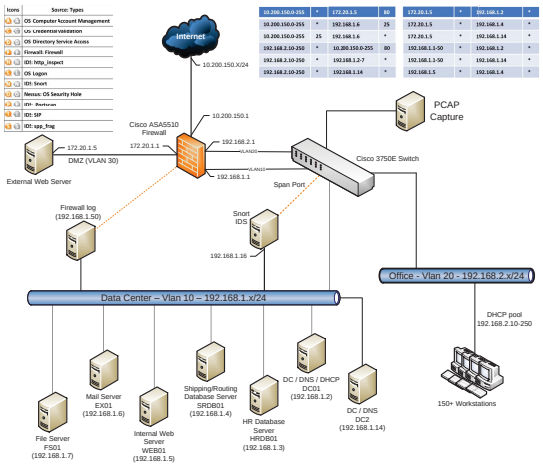


Figure 6: AFC network topology. Acceptable flow rules are listed on the top right. Anomaly icons representing incidents from IDS, firewall, system and Nessus logs are depicted on the top left.

This approach works for all the layout changes in the compressed graph interactions.

Staged animations are designed upon the transition between graph views. In the first stage, the single/mega-node not in the current graph will move to the initial position of a new mega-node in the current graph to which this single/mega-node belongs. In case it belongs to no one in the current graph, it just disappears in the first stage. In the second stage, the nodes new in the current graph appear in their initial positions as computed. In the last stage, all the nodes in the current graph move from their initial positions to the optimal positions according to the layout algorithm.

6 Case Study

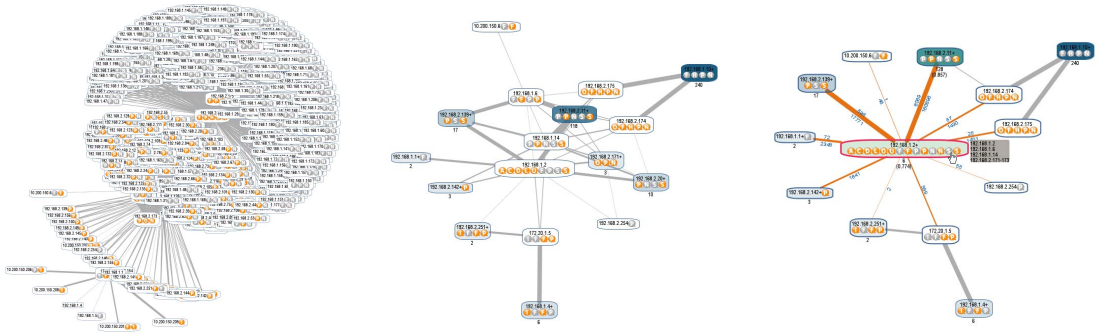
6.1 Network Situation Awareness

We evaluate our graph visualization tool with CNG on the VAST 2011 Mini Challenge-II dataset [1]. The challenge dataset includes a computer network architecture (Figure 6) of a shipping company - All Freight Corporation (AFC), its security policy rules, and about three days of monitoring reports from the corporate network, which consist of a firewall log, an intrusion detection system (IDS) log, an aggregated file of system logs, and a Nessus network vulnerability scan report.

Network traffic graphs are constructed by aggregating the network flows recorded in the firewall and IDS logs. Each flow is represented by the IP address of the source and destination host, as well as the flow start/end timestamps. Note that we drop the host port number and use IP address only to reduce the graph size. The raw flow entries are further segmented by time and placed into several bins based on a time window setting (an hour by default) to create a graph for every time window. The network graph of any consecutive time windows can be generated by merging the per-window graphs.

Over the graphs, we attach the anomalies pre-computed from the data to speedup the human analysis. For the firewall logs, we translate from the AFC's security policy rules into the Acceptable Flow Rules (AFR) (Figure 6) and detect firewall anomalies by filtering the firewall log with the AFR. Other types of anomalies are generated by parsing the IDS (snort), Nessus, and system logs. Figure 6 lists the type and source of anomalies we detect in this case and also the anomaly icons representing them in the graph. Note that, for an anomaly on a flow (e.g. firewall and IDS), we further split it into anomalies on both of its endpoints. The icon colors are used to differentiate the abnormal flow source (orange) and destination (grey).

Below we give a detailed user trail to show case how our tool



(a) Original (b) Compressed (c) Compressed with similarity = 0.5

Figure 7: Overview traffic graphs of AFC corporate network with different settings.

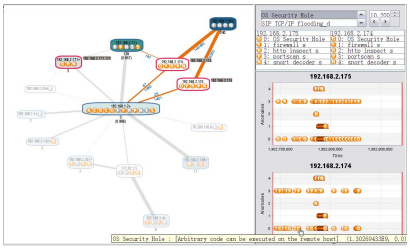


Figure 8: The hosts with security holes and the cross-subnet port scans from 192.168.2.174/175.

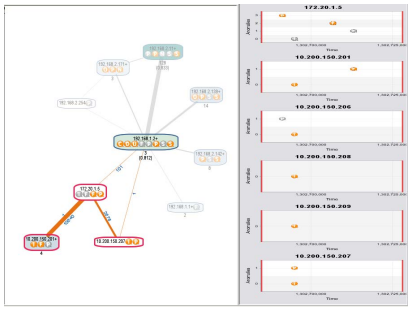


Figure 9: DoS attacks against the corporate web server (172.20.1.5) from 10.200.150.201, 206-209 indicated by the anomaly icon (I).

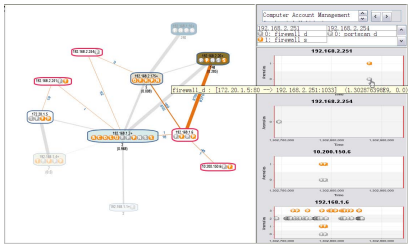


Figure 10: Email exchanges with an external host (10.200.150.6) and the undocumented computers (192.168.2.251/254).

helps to increase situation awareness and detects malicious attacks. Consider John, the computer network operation lead of AFC, is checking the corporate network status of the recent three days for noteworthy events. He starts by loading the whole network graph in this period, as shown in Figure 7(a). Because the view is too messy, he continues by applying the basic CNG to create a compressed network graph, as shown in Figure 7(b). From this graph, he quickly learns some key hosts in the period (e.g. 1.2, 1.14. "192.168." is omitted throughout this study), but still feel a little prohibitive to proceed to details. He decides to further simplify the graph by using the fuzzy CNG with a similarity score of 0.5. The resulting graph in Figure 7(c) is clear enough for his overview purpose: the hosts in the central fuzzy node group (1.2, 1.6, 1.14 and 2.171-173) and 2.174, 2.175 are all of the hub nodes in the graph.

Port Scan & OS Security Holes.

Based on the AFC network structure (Figure 6), John bypasses three server machines (1.2, 1.6, 1.14) which routinely communicate with all the hosts for DNS/data services. Also in his knowledge, the suspicious behaviors of a hub node, e.g. port scan, often associate with the OS security holes. So he clicks on the "OS security hole" anomaly type and highlights all the hosts with such anomaly on the graph. To drill-down to the individual hosts, he splits the fuzzy group and locates 2.171-2.175 as the threats. He finds that 2.174 and 2.175 are more dangerous because they have a higher port-scan rate (by the link thickness) and also initiate cross-subnet floods to 1.10-250 where many hosts do not have physical machines. The screenshot with the temporal anomaly view of 2.174 and 2.175 is given in the right panel of Figure 8.

DoS Attacks

A critical server John examines in the following is the AFC's external web server (172.20.1.5). With the CNG-compressed graph, the web server is easily found as it has outstanding connection patterns from the other hosts. A single click on the node shows up a noteworthy anomaly icons (I) on the morning of the first day, suggesting that there could be Denial-of-Service (DoS) attacks at that time. John further drills down to that period with the time range selector and highlight the web server's egocentric traffic graph. Figure 9 confirms the potential DDoS attacks from the external hosts 10.200.150.201, 206-209, due to the anomalies happened simultaneously with the web server.

Social Engineering and Undocumented Computer

Based on the existing inputs from the visualization, John is able to manually construct a graph by combining the CNG hints and the subnet attribute of the hosts. As given in Figure 10, the semi-automatically compressed graph shows the connections among hosts with different usages/subnets. John keeps the mail server (1.6) standalone, as he wants to check the email exchanges between AFC and outside. Clearly, an external host 10.100.150.6 have a bi-directional email exchanges with the AFC mail server, as indicated in the temporal anomaly view. This finding can be examined further as a clue for potential social engineering attacks.

As indicated in Figure 6, the AFC workstations have an IP range from 192.168.2.10 to 192.168.2.250. However, in the compressed graph (Figure 10), two new IPs (2.251 and 2.254) appear that is outside the IP range. What more suspicious is the host 2.251, which even has a two-way communications with the external web server at the end of the period.

Note that in our future plan, the tool will support streaming network traffic, then John will be able to access the real-time traffic patterns and increase the timeliness in detecting the events.

6.2 Honeypot Monitoring

Another security-related scenario our tool works well is the Honeypot monitoring. In this case, a networked computer, called the hon-

eypot, is deployed in the corporate network or the Internet to mitigate potential threats to the organization's network and to collect the attacker's behaviors and patterns for further security researches. The honeypot data set we worked on comes from the VizSec community [2] and is provided in [31]. The dataset contains 14 million labeled flows and 7 million alerts from a single honeypot lasting about six days. Flows can be broken down into types of SSH, FTP, HTTP, AUTH/IDENT, IRC and OTHERS. The traffic data is preprocessed by making SQL queries linking flow and alert tables and sorting the output into bins based on the time window (e.g., one hour). The connections are finally transformed into graphs together with alerts attached on the nodes. The alert icon/type pairs in this data are "A"/AUTH, "F"/FTP, "I"/ICMP, "R"/IRC, "S"/SSH.

The network administrator or researcher can visualize the honeypot-centric network using our tool. Take a seven-hour trace as an example, the initial view (Figure 11(a)) is overwhelmed by the connected hosts. After the compression with the directed graph CNG, a very clear graph shows up (Figure 11(b)), which contains only three nodes: 146.217.254.148 (the honeypot), 103.53.0.211+ (9 hosts that the honeypot scans without responses), and 187.79.2.4+ (507 hosts the honeypot communicates bidirectionally). By setting the link aggregation to the "average" mode, it can be found that the 507 hosts have a larger "to honeypot" traffic than the "from" traffic (potential attackers), while the 9 hosts merely set up one flow with the honeypot in average. To access dynamic patterns of the attackers, the user can check the "temporal" option and generate a compressed version of the dynamic graph, which exploits different groups of temporal connections with the honeypot (Figure 11(c)). A group of nodes in this graph, 187.79.2.4+ (206 hosts), show an abnormal traffic burst in the beginning of the time period. Combined with the outstanding "F"/FTP icon on the node, it suggests that the honeypot has been compromised early in this period by these attackers, and numerous FTP attacks to subnets of IP prefixes are tried with weak SSH passwords. Further splitting and re-compressing this node group with both the "weighted" and "temporal" options reveals the suspected FTP attackers (the red nodes in Figure 11(d)) with a significant traffic volume to the honeypot.

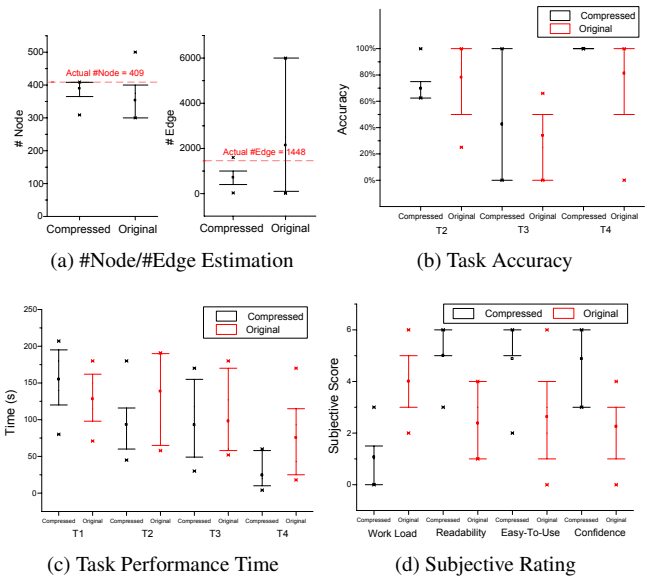
6.3 Data Center

In this case, we have tried our technique on the traffic flow graph among data centers. The traces are collected from a large corporate in the typical NetFlow format, containing statistics of the flows, i.e., timestamp, flow sequence, src/dst IPs and ports, duration, packets, flags, etc. The original data was divided into bins of 15-minute time window for analysis.

Figure 12(a) shows the original flow graph with 6509 nodes and 18347 edges. For a cleaner visualization of the overall topology, a smaller graph containing only 50 nodes are displayed through the node filtering (Figure 12(b)). Nevertheless, The topology is preserved much better with node filtering over the compressed graph (Figure 12(c)) than directly on the original graph.

6.4 Social Network

We also experiment CNG on the widespread social network graphs. The dataset we apply is from KDD Cup 2012 Track-I [3], which is about the social networks on Tencent Weibo, one of the largest micro-blogging websites in China. There are 4710 unique items (person, organizations or groups) being recommended to 1392873 unique users, who either accept or reject the recommendations. We select the top K (K=10) popular items accepted by the users as well as the top K popular users followed by the other users. For each popular item and user, we generate the below graphs: 1) User following graph; 2) Interaction (tweet) graph; 3) Interaction (retweet) graph; and 4) Interaction (comment) graph. The CNG performance on these social graphs are given in Table 4. While CNG does perform good in the running time scaling to very large graphs, the results in terms of the compression rate are not satisfying in this case, suggesting the future works in dealing with the small-world graphs.



(a) #Node/#Edge Estimation (b) Task Accuracy

(c) Task Performance Time (d) Subjective Rating

Figure 13: User experiment results showing the median, 5th/95th and 1st/99th percentile.

7 User Experiment

We conducted an experiment to compare the graph visualization tool with CNG (compressed) and without CNG (original). Participants were asked to perform several graph analysis tasks on the VAST Challenge 2011 dataset with both tools. We measured the performance time for each task, and the accuracy. Following the tasks with each tool, participants responded to a quantitative questionnaire regarding their experience and also provided verbal feedbacks.

Task

Four tasks are designed on the traffic graph of the AFC corporate network.

T1: Estimate the number of servers/hosts in the network and the number of distinct flows between pairs of servers/hosts. This is to evaluate the user's basic understanding of the graph scale according to the tool.

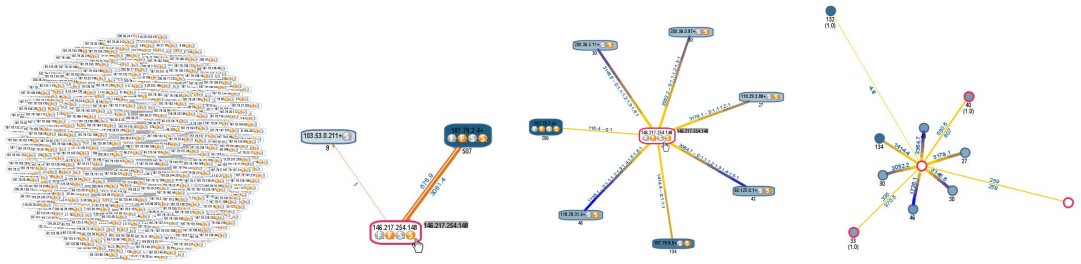
T2: Find out all the hub servers/hosts in the network with more than 100 connections. The hub nodes are the basis to understand the graph topology.

T3: Find out all the hub servers/hosts in the network suspected to have illegal behaviors (e.g. malicious port-scan). This is to evaluate the ability to combine the graph topology with the content (IP address) for the analysis.

T4: Find out a server/host external to the AFC network and ever has email exchanges with the AFC host machines. This is to evaluate the performance in checking details with the graph.

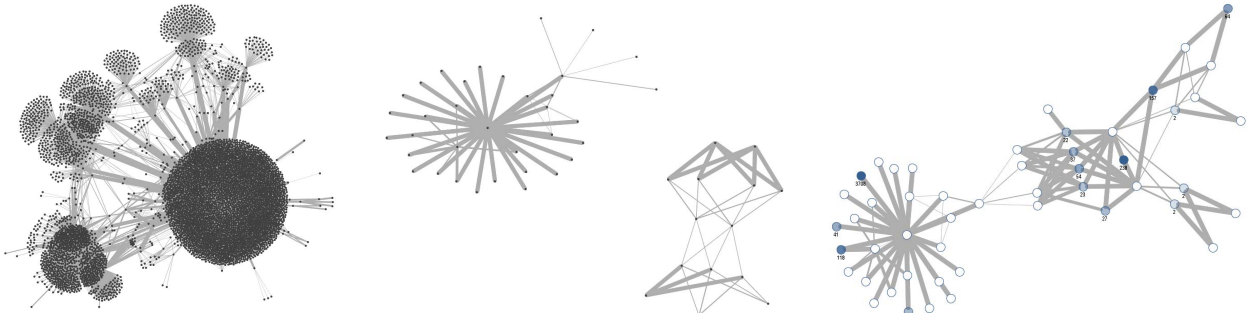
Design

Our experiment follows the within-subject design principle. Each participate performs all the tasks with each of the tool. To avoid the learning and ordering effects, we map the key IP addresses in the raw VAST Challenge data into a new set of IPs to create two similar data sets. The users are partitioned into four groups with different combinations of the tool usage order and the data set. Before the user performs the task with each tool, a training session is also conducted with an irrelevant sample data on the basic visual encoding and interaction methods of the tool. Participants are told to complete the tasks without any pause in a best-effort manner. Approximately 3 minutes are given as a deadline for each task.



(a) Original (b) Directed CNG (c) Dynamic CNG (d) Dynamic&Weighted CNG

Figure 11: CNG-compressed graph visualization for a single honeypot's view.



(a) Original (b) Filter to 50 nodes from the original graph (c) Filter to 50 nodes from the compressed graph

Figure 12: Data center flow graph visualization in the original tool, with the node filter and after integrating the CNG method.

Apparatus

The experiment was ran on a standard laptop with a 2.4 GHz Intel Core 2 Duo processor, 3GB memory, a 1440 x 900 15 widescreen LCD display, and an optical mouse. The software was written in Java 1.6 and ran from the Windows XP command line.

Participant

Eight participants (7 male, 1 female), with ages ranging from 23 to 36 years, attended the experiment. All of them are technical persons with a computer science background. A half even have experiences with network/system administration, which accounts for the bias of the gender distribution. We select this sample with more professionals because the graph visualization tool in this scenario mainly targets for the network administrators with rich experiences.

Result

The experiment results are summarized in Figure 13.

Accuracy: As in Figure 13(a), on the estimation of graph nodes, the tool with CNG performs much better. An analysis of variance (ANOVA) test shows a significant difference on the error rate ($F(1,14) = 9.72$, $p < .01$). On the graph edges, CNG has a lower error rate in average, but not significant enough. Note that the compressed graph has a tendency to underestimate the number of edges for user, while the original graph has the opposite tendency. In the other tasks, the accuracy rates are not significantly different (Figure 13(b)), although on T2 and T4 where most users can work out the correct answer, the accuracy rate variances of CNG are much smaller.

Performance Time: CNG spends slightly longer time in completing the node/edge estimation task (Figure 13(c)), but leads to a more accurate result. On the performance times of T2 to T4, still similar to the accuracy rate, Users spend shorter time on T2 and T4 with CNG, with a significance on T4 ($F(1,14) = 6.85839$, $p < .05$).

In summary, the graph visualization with CNG shows advantage over the original one in estimating the graph scale, especially on the number of nodes. The accuracy and performance time are also better with CNG in understanding the topology and accessing the details. Specially, CNG is significantly quicker in drilling down to the details. On the other hand, both the tools do not perform well in combining the graph topology with the subject content on nodes/edges, which suggests a future improvement on the graph attribute visual analysis.

Subjective Feedback and Observation

Surprisingly, the subjective feedbacks from the users demonstrate an unanimous favor towards CNG (Figure 13(d)). The ANOVA test shows a significant difference ($F(1,14) > 20$, $p < .001$) on the work load and readability; and a less significant difference ($F(1,14) > 8$, $p < .01$) on the easy-to-use and confidence. The users stated the CNG tool as "simple to use", "low amount of work", "the highest amount of confidence", "pretty good readability", although "not fully automated", "require some efforts as the first time user". In comparison, the original tool is stated as "kind of pain to use", "not readable because it's cluttered", "no confidence being able to pick up anything other than wide guess", "random, probably bad for large data". A user even stated for the CNG: "In terms of graphical tools, this is cleaner than many commercial tools that I have used such as HP Openview". We also observed an interesting fact that when the users work with the original graph, they use a lot of interactions, e.g. zoom & pan, trying to understand the graph. While working with the CNG-compressed graph, few users move the graph greatly. As the user said, it already gives a near perfect summarization.

8 Conclusion

In this paper, we propose the Collaborative Node Grouping (CNG) method to visually condense the huge graph without sacrificing any graph information. It is shown that CNG can effectively reduce the scale of many real-world graphs and still preserve critical features of the original graph. The classical node-link representation is introduced to visualize the CNG-compressed graph, with carefully designed visual encoding, stable layout and animated transition to provide the user with ease and interactivity in analyzing with the compressed graph. We showcase the application of the graph visualization tool with CNG in four use cases, as well as the usage of CNG extensions to the directed, weighted and dynamic graphs. A controlled user study on the effectiveness of CNG suggests that a lot of domain users can benefit from the method and are more enjoyable in the graph overview and detail analysis tasks with CNG.

REFERENCES

[1] IEEE VAST Challenge, 2011. http://hcil.cs.umd.edu/localphp/hcil/vast11/.

[2] VizSec community, 2011. http://vizsec.org.

[3] KDD Cup 2012: user modeling based on microblog data and search click data, 2012. http://www.kddcup2012.org/.

[4] J. Abello, F. van Ham, and N. Krishnan. ASK-GraphView: A large scale graph visualization system. In *IEEE Symposium on Information Visualization (InfoVis'06)*, 2006.

[5] D. Auber, Y. Chiricota, F. Jourdan, and G. Melancon. Multiscale visualization of small world networks. In *IEEE Symposium on Information Visualization*, 2003.

[6] S. T. Barnard and H. D. Simon. A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems. *Concurrency Practice and Experience*, 6(2):101–117, 1994.

[7] A. Broder, M. Charikar, A. Frieze, and M. Mitzenmacher. Min-wise independent permutations. *Journal of Computer and System Sciences*, 60:630–659, 2000.

[8] F. Chierichetti, R. Kumar, and S. Lattanzi. On compressing social networks. In *ACM SIGKDD conference on Knowledge Discovery and Data Mining*, 2009.

[9] W. Cui, H. Zhou, H. Qu, P. C. Wong, and X. Li. Geometry-based edge clustering for graph visualization. In *IEEE Symposium on Information Visualization*, 2008.

[10] T. A. Davis and Y. Hu. The university of florida sparse matrix collection. *ACM Transactions on Mathematical Software*, 38(1), 2009.

[11] V. de Silva and J. Tenenbaum. Global versus local methods in nonlinear dimensionality reduction. In *Annual Conference on Neural Information Processing Systems*, 2002.

[12] P. Eades. A heuristic for graph drawing. In *Congressus Numerantium*, volume 42, pages 149–160, 1984.

[13] T. M. J. Fruchterman and E. M. Reingold. Graph drawing by force-directed placement. *Software, Practice & Experience*, 21(11):1129–1164, 1991.

[14] P. Gajer and S. G. Kobourov. GRIP: Graph drawing with intelligent placement. *Journal of Graph Algorithms and Applications*, 6(3):203–224, 2002.

[15] E. Gansner, Y. Koren, and S. North. Topological fisheye views for visualizing large graphs. In *IEEE Symposium on Information Visualization (InfoVis'04)*, 2004.

[16] E. R. Gansner, Y. Koren, and S. North. Graph drawing by stress majorization. In *International Symposium on Graph Drawing*, 2004.

[17] D. Harel and Y. Koren. Graph drawing by high-dimensional embedding. *Journal of Graph Algorithms and Applications*, 8(2):195–214, 2004.

[18] B. Hendrickson and R. Leland. A multilevel algorithm for partitioning graphs. In *ACM/IEEE conference on Supercomputing*, 1995.

[19] N. Henry and J.-D. Fekete. MatrixExplorer: a dual-representation system to explore social networks. In *IEEE Symposium on Information Visualization (InfoVis'06)*, 2006.

[20] N. Henry, J.-D. Fekete, and M. J. McGuffin. NodeTrix: a hybrid visualization of social networks. In *IEEE Symposium on Information Visualization (InfoVis'07)*, 2007.

[21] D. Holten. Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. In *IEEE Symposium on Information Visualization (InfoVis'06)*, 2006.

[22] Y. Hu. Efficient and high quality force-directed graph drawing. *Mathematica Journal*, 10(1):37–71, 2005.

[23] Y. Hu. *Combinatorial Scientific Computing*, chapter Algorithms for Visualizing Large Networks, pages 525–549. CRC Press, 2012.

[24] Y. Jia, J. Hoberock, M. Garland, and J. C. Hart. On the visualization of social and other scale-free networks. In *IEEE Symposium on Information Visualization*, 2008.

[25] T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Inf. Process. Lett.*, 31(1):7–15, 1989.

[26] Y. Koren, L. Carmel, and D. Harel. ACE: A fast multiscale eigenvector computation for drawing huge graphs. In *Infovis '02*, 2002.

[27] G. Kumar and M. Garland. Visual exploration of complex time-varying graphs. In *IEEE Symposium on Information Visualization (InfoVis'06)*, 2006.

[28] J. Lamping, R. Rao, and P. Pirolli. A focus+context technique based on hyperbolic geometry for visualizing large hierarchies. In *CHI*, 1995.

[29] T. Munzner. H3: Laying out large directed graphs in 3d hyperbolic space. In *IEEE Symposium on Information Visualization (InfoVis'97)*, 1997.

[30] A. Quigley and P. Eades. FADE: Graph drawing, clustering and visual abstraction. In *International Symposium on Graph Drawing*, 2000.

[31] A. Sperotto, R. Sadre, F. van Vliet, and A. Pras. A labeled data set for flow-based intrusion detection. In *Proceedings of the 9th IEEE International Workshop on IP Operations and Management (IPOM '09)*, pages 39–50, Berlin, Heidelberg, 2009.

[32] F. van Ham and A. Perer. "search, show context, expand on demand": Supporting large graph exploration with degree-of-interest. In *IEEE Symposium on Information Visualization (InfoVis'09)*, 2009.

[33] F. van Ham and M. Wattenberg. Centrality based visualization of small world graphs. In *EuroVis*, 2008.

[34] M. Wattenberg. Visual exploration of multivariate graphs. In *SIGCHI conference on Human Factors in computing systems (CHI'06)*, 2006.