Multitree-like Graph Layering Crossing Optimization

Radek Mařík
Faculty of Electrical Engineering, Czech Technical University, Technicka 2, Prague, Czech Republic

Keywords: Crossing Optimization, Layered Graph, Multitree, Spanning Tree, Phylogenetic Network, Genealogical Network.

Abstract: We improve a method of multitree-like graph visualization using a spanning tree-driven layout technique with constraints specified by layers and the ordering of groups of nodes within layers. We propose a new method of how the order of subtrees selected by the driving spanning tree can be derived from the actual edge crossings. Such a subtree order leads to additional decreasing of total edge crossings from 1% to 50%. This depends on the shape of the processed graph, ranging from a pure tree to a general acyclic graph. Our achievements are demonstrated using several datasets containing up to millions of people, species, or services. The proposed subtree ordering method of layered graphs that are similar to acyclic multitrees retains the generating of acceptable layouts in almost linear time.

1 INTRODUCTION

Some applications lead to causality driven networks represented as acyclic graphs. If a kind of inheritance is involved, then we often deal with so-called multitrees. We use a genealogical graph as an example of general multitree-like networks. However, similar results can be demonstrated in other domains, such as telecommunications services and phylogenetic graphs. In this paper, we focus on methods that are capable of visualizing whole societies with millions of nodes in which the layouts enable an assessment of general global trends and related features.

Graph visualization technique research remains a highly popular field, having attracted much attention for decades (Tutte, 1963; Gibson et al., 2013). Tree based drawing methods of phylogenetic/genealogical graphs have been among the standard techniques for centuries. Present software implementations often layer nodes as proposed by various authors 20 years ago (Sugiyama et al., 1981; Gansner et al., 1993; Gansner and North, 2000; Graphviz, 2016). In some cases, it is necessary to assess top-level structures of the entire network in order to select the appropriate subsequent processing steps. These cases lead to a requirement to display the entire network of families, or at least a significant part, in one layout. However, a majority of algorithms contain processing steps with an asymptotic complexity higher than the linear one. Such implementations are often not capable of coping with graphs over 100,000 nodes. We also face other issues with challenges linked with edge crossing and preferences on node clustering (Warfield, 1977; Sugiyama et al., 1981; Sugiyama and Misue, 1991). Therefore, the standard techniques for planar graph layouts (an ideal layout example with no edge crossings) (Lempel et al., 1967; Hopcroft and Tarjan, 1974; Booth and Lueker, 1976; Shih and Hsu, 1999; Hsu and McConnell, 2004; Reingold and Tilford, 1981) including planarization techniques (Resende and Ribeiro, 2001; Chimani et al., 2008; Chimani et al., 2011; Mathews and Frey, 2012) are not suitable in all instances.

As we adopt multitree-like networks, we stress the significance of layers, so we consider a layout design targeting layered drawing (Healy and Nikolov, 2013). The majority of algorithms that compute layers are derived from topological order computation, $O(|V|+|E|)$ time complexity (Cormen et al., 2009). The algorithms choose one of many possible solutions that satisfy layer intervals of node placements. In this paper, we also focus on techniques with linear asymptotic complexity. Furthermore, our approach enables the possibility to group nodes assigned to the same layer while keeping edge crossing minimized. The underlying assumption relies on the proximity of the processed graph to the multi-tree form.

Our approach follows the general framework consisting of four steps proposed by Sugiyama (Sugiyama et al., 1981). However,

Mařík, R.
Multitree-like Graph Layering Crossing Optimization.
DOI: 10.5220/0007345302330240
In *Proceedings of the 14th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP 2019)*, pages 233-240
ISBN: 978-989-758-354-4

each step could be accomplished using different techniques. In (Marik, 2016) and (Marik, 2017a), a new method targeting multitree-like networks that allows the determination of node order constraints within layers using an undirected spanning tree-driven layout of subtrees is proposed. The spanning tree controls a selection of subtrees and their ordering during the layout process. In (Marik, 2017a), the order is determined based on subtree order (the number of nodes) that could be a too rough estimate of real edge crossings.

In this paper, we focus on the optimization of subtree selection to further reduce edge crossings. The rest of the steps, e.g. spanning tree selection, node ordering and node positioning, follow the methods proposed in (Marik, 2017a; Marik, 2017b; Marik, 2018).

In summary, we improve the fourth step of the method proposed in (Marik, 2017a), Section 3, node ordering within layers using a different subtree selection criterion. Thus, we treat the most critical aspect discussed in (Sugiyama et al., 1981), and particularly address the second step of the main algorithm proposed in (Gansner et al., 1993):

1. determination of generations (layers),
2. enforcing node orders within the layers,
3. setting the actual layout coordinates of nodes,
4. design of edges.

The remainder of the paper is organized as follows: Section 2 provides an overview of methods related to the layering step. To create the appropriate context for the proposed algorithms, we also summarize the steps of approaches described in several other papers (Marik, 2017a; Marik, 2017b; Marik, 2018). All steps can be accomplished using an almost linear algorithm within the framework that relies on multitree properties. In Section 3, we provide two algorithms that achieve improved total edge crossings using actual edge crossings computed for every subtree of the driving spanning tree. Finally in Section 4, we discuss achieved results tested on datasets with up to 10^6 nodes.

2 RELATED METHODS

In this section, we provide a brief overview of the methods related to those proposed in this article. In fact, we provide a brief overview of the steps of Sugiyama's framework. We describe related methods for each step and also the technique we currently use in the approach focused on multitree-like networks.

We follow the usual graph theory terminology (Diestel, 2005; Bondy and Murty, 2008; Wilson, 1998). In the case of family trees, we assume that children are not linked directly to their parents, but through so-called marriage nodes (V_M). Each marriage node represents a marriage in which children were born. Further, we aim for a layout in which children linked to the same marriage node are assigned to the same layer and grouped.

We use the following terms. A **layering** $L = (L_1, L_2, \ldots, L_h)$ of a graph, $G = (V, E)$ is an ordered partition of V into non-empty *layers* L_i such that adjacent nodes are in different layers, i.e. if $(u, v) \in E$, where $u \in L_i$ and $v \in L_j$, then $i \neq j$ (Brandes and Köpf, 2002; Healy and Nikolov, 2003; Nikolov et al., 2005; Lutteropp, 2014). Let $L(v) = i$ if $v \in L_i$. The index i is called *node layer (rank)*. Regardless whether G is directed or undirected, an edge incident to $u, v \in V$ is denoted by (u, v) if $L(u) < L(v)$. An edge (u, v) is *short* if $L(v) - L(u) = 1$, otherwise it is *long* and *spans* layers $L_{L(u)+1}, \ldots, L_{L(v)-1}$. Let $N_v^- = \{u : (u, v) \in E\}$ ($N_v^+ = \{w : (v, w) \in E\}$) denote the *upper* (*lower*) *neighbors* and $d_v^- = |N_v^-|$ ($d_v^+ = |N_v^+|$) the *upper* (*lower*) *degree* of $v \in V$. The *height* h is the number of layers, and the *width* is the number of nodes in the largest layer. The *span of an edge* is the difference between the layers of the nodes to which it is incident. A digraph is proper if no edge has a span greater than 1. A *layered graph* $G = (V, E; L)$ is a graph G together with a layering L.

The **ordering** of a layered graph is a partial order $\prec$ of V such that either $u \prec v$ or $v \prec u$ if and only if $L(u) = L(v)$ (Brandes and Köpf, 2002). We denote $v_j^{(i)} \in L_i$ where $L_i = \{v_1^{(i)}, \ldots, v_{|L_i|}^{(i)}\}$ with $v_1^{(i)} \prec \cdots \prec v_{|L_i|}^{(i)}$. The *position* $\mathrm{pos}[v_j^{(i)}] = j$ and the *predecessor* of $v_j^{(i)}$ with $j > 1$ is $\mathrm{pred}[v_j^{(i)}] = v_{j-1}^{(i)}$. An edge segment (u, v) is said to *cross* an edge segment (u', v'), if $u, u' \in L_i$, $v, v' \in L_{i+1}$, and either $u \prec u'$ and $v' \prec v$, or $u' \prec u$ and $v \prec v'$.

2.1 Network Components

The majority of real-life datasets consist of several or many disjoint connected components. It depends on the given task whether one requires the processing of all components or a specific one. For the purposes of this paper, we always selected a connected component that has a maximum number of nodes.

2.2 Treatment of Strongly Connected Components

An input dataset might capture a network with cycles. A number of efficient algorithms are based on properties of DFS. If a processed graph contains cycles, then

some algorithms might fail, such as a topological order computation, or they might generate unnaturally long paths. Therefore we need to break the cycles in each strongly connected component (SCC).

In this paper, we only confirm that the network is acyclic and we perform a simple non-optimal algorithm to remove cycles in the unlikely event that some are found. We remove all loops and back edges of a randomly selected DFS-tree from the given SCC in the dataset experiments performed in this paper if such an SCC was detected. For the remainder of the paper, we assume that the processed network is an acyclic graph.

2.3 Node Layering

Assuming that the processed network is acyclic, the layout design continues with node layering in the next step as proposed in (Sugiyama et al., 1981; Gansner et al., 1993; Gansner and North, 2000). In this paper, we utilized a solution to the layering problem that guarantees additional domain layer constraints, such as the layering of siblings in family trees, as proposed in (Marik, 2017a; Marik, 2017b). The solution relies on a driving spanning tree that controls the node ordering design (Marik, 2017a), i.e. the nodes are processed in the order resulting from the searching of the spanning tree when the next node is chosen based on an edge crossings estimation criterion. The spanning tree selection using a graph block analysis in which cases of non-trivial blocks are resolved using integer linear programming (ILP) was presented in (Marik, 2017b). It identifies a spanning tree that minimizes the span over all layers of the given block. The blocks can be identified in linear time (Paton, 1971; Hopcroft and Tarjan, 1973). A proper minimum spanning tree for each block consists only of its short edges. As blocks with a structure more complex than a single undirected cycle are very rare in real networks or they are very small, the time complexity of the driving spanning tree selection still remains practically linear.

2.4 Design of Node Order within Layers

Node ordering on each layer is often designed as permutations of the vertices within a given layer, leading to the minimum number of edge crossings (Warfield, 1977; Sugiyama et al., 1981; Gansner et al., 1993). However, these traditional methods do not treat apriori node order constraints such as node grouping. When dealing with genealogical graphs, the order of node subgroups needs to be satisfied. For example, the order of siblings is often defined by their birth dates and the sibling sequences should not be interrupted by other nodes. Then, we deal with a constrained crossing reduction problem. Early formulas for the computation of the number of crossings can be tracked to (Warfield, 1977). The problem is known to be NP-hard (Eades et al., 1986). There are many heuristics for edge crossings reduction. A simple heuristic for the one-sided two-level crossing reduction can be based on barycenter values (Forster, 2005), but the algorithm implementing the crossing reduction problem given constraints on nodes still runs in quadratic time.

Having a driving spanning tree, node layering can be performed as proposed in (Marik, 2017a). In fact, we use the same algorithm in this paper, but the criterion sorting the subtrees of a current given root node based on edge crossings is evaluated differently.

Let us outline the critical steps of the algorithm. Starting from the node with the lowest layer we assign nodes of subtrees into layer arrays (initially empty for each layer), see Fig. 1 for more details. First, subtrees with a minimum layer higher than the layer of the current node v_i are processed because their edges do not cross any other edges in the rest of the graph. Then the remaining subtrees are processed according to their increasing edge crossings of edges between nodes at the current node layer and the successor node layer. This processing order can be justified as follows (Marik, 2017a). Let us assume we process the current node v_i where some children's nodes link K subtrees with a minimum layer lower than the current node layer. A sequence $[cr_1, \dots cr_K]$ is obtained if the subtrees are sorted according to their edge crossing counts cr_ℓ between the children's node and its subtree. If these subtrees are layered side by side and each child is linked with them, then the total number of injected edge crossings is $CR_{v_i} = \sum_{j=2}^{K} \sum_{k=1}^{j-1} cr_k = \sum_{\ell=1}^{K-1} (K-\ell) cr_\ell$ that is the minimum if the sequence $[cr_1, \dots cr_K]$ is not decreasing.

In (Marik, 2017a) the edge crossings for each subtree was estimated using subtree orders (the number of nodes) because they can be computed in linear time for all subtrees of the spanning tree.

Indeed, we focus on this step of node ordering within layers in this paper. We show that the number of edge crossings itself for each subtree does not need to be estimated because it can be determined directly and in a still efficient, almost linear way.

2.5 Node Positioning

Node positioning is completed in the final step of the entire layout process. Node ordering within layers, which we cover in this paper, does not depend on this

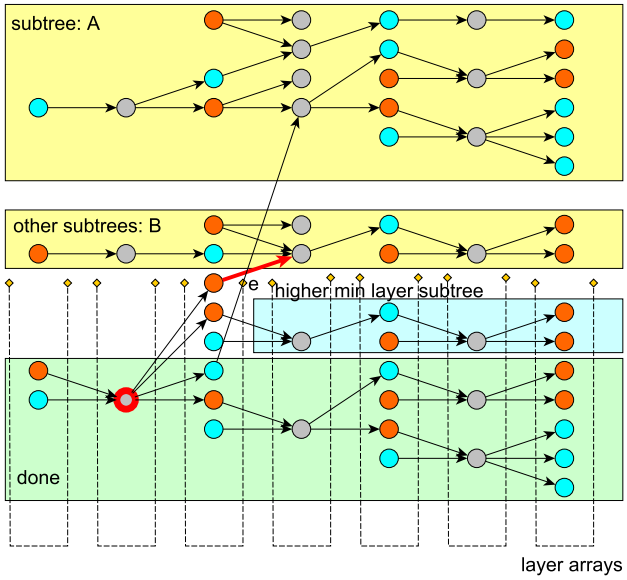

Figure 1: A symbolical snapshot of the layout method proposed in (Marik, 2017a). The current node is identified by its thick red border. Blue nodes represent men, orange nodes represent women, gray nodes represent marriages. The greenish zone is an already processed part of the graph with all nodes registered in the layer arrays that keep their order of registration. The blue zone contains just one subtree with the minimum layer higher than the layer of the current node. Two yellow zones represent two other subtrees layered in the order based on the used criterion estimating the number of edge crossings.

step. In (Marik, 2018) the node positioning method based on the force-driven approach with barrier-like repulsive forces that keeps the order of nodes within layers and avoids the quadratic complexity of traditional methods was proposed. The force-directed based method positions the ordered nodes in layers in almost linear time. This method was also used in the experiments discussed in this paper.

3 SUBTREE EDGE CROSSINGS

In this section, we propose a new method that can be used to calculate the number of edge crossings of any subtree at its root layer given an acyclic layered graph and its undirected driving spanning tree minimizing the number of graph layers. As we mentioned in Section 2.3, such numbers of edge crossings should be used in the subtree ordering criterion that influences the total edge crossings in the resulting graph layout.

Let us summarize some constraints dealing with the driving spanning tree and its subtrees in the process of node ordering within layers. In this usage we assume that the inputed acyclic layered graph is proper, i.e. long edges are replaced by a simple sequence of short edges spanning the same layers. The spanning tree is searched through from its leaf node. At every step r, an edge $e^r \in \{e_i^r\}_{i=1}^K$ from all edges e_i^r incident with the current node is selected and the subtree determined by the edge is recursively added. Thus, nodes of subtrees are added one by one to the layer arrays representing the sequences of nodes within layers. Although we refer to a subtree T_i, the algorithm operates only with its root node n_i^r.

To ensure a low number of edge crossings an edge e_j^r incident with the subtree T_j having the least number of edge crossings K_j is selected. See Fig. 1, where the edge e^r is highlighted as the red edge e incident with the subtree B while another edge incident with the subtree A generates three edge crossings inside the subtree B. The number of edge crossings cr_i is computed as the number of all edges of the subtree T_i spanning the same layers as the edge e_i^r, i.e. this number of edges must be crossed by edges leading to other subtrees if the subtree T_i is processed before than the other subtrees.

The basic variant of an algorithm that computes edge crossing cr_i for all possible subtrees of the selected driving undirected spanning tree is rather simple. Initially, one notices that any such subtree is determined by its root node and one of the edges incident with the root. In other words, each edge of the spanning tree determines two subtrees of the spanning tree with the edge end nodes as their root nodes. Then, the number of edge crossings cr_i can be counted in the following way:

1. for all edges e_i of the spanning tree
   (a) remove the edge e_i from the spanning tree

(b) for both the resulting subtrees T_j
   i. count the number cr_j of edges of the subtree spanning the same layers as the edge e_i.

The algorithm repeats the inner tree search for each edge of the spanning tree, i.e. with the complexity $O(N)$, where N is the number of nodes. The inner tree search can also be performed in linear time $O(N)$. Therefore, the resulting asymptotic complexity is $O(N^2)$. As this is the only step in the proposed layout design with the quadratic asymptotic complexity, it significantly constrains volumes of networks that might be processed up to 10^5 nodes using the current experimental Python implementation.

However, assuming a number of layers significantly lower than the number of network nodes, e.g. 10^2 layers for networks with 10^6 nodes, it is possible to use the following almost linear algorithm. Its idea is based on two facts. Initially, the sum of counts of edges spanning any pair of layers of two subtrees incident to any given edge including the edge itself is constant and equal to the edge count spanning the same pairs of layers calculated for the entire graph. Secondly, a similar property is held for any node. That means the sum of counts of edges spanning any pair of layers of all subtrees incident to edges incident with the given node including also these edges is constant and equal to the edge count spanning the same pairs of layers calculated for the entire graph. Thus, if subtrees (i.e. their root nodes) are processed in a postorder sequence, the edge counts for all layers can be propagated from leaves of the driving spanning tree. If a node is not a leaf then counts of edges spanning two layers of its already processed subtrees can be combined and edge counts related to the only uncovered edge calculated. Thus, this algorithm uses only a single DFS scan through the driving spanning tree with linear complexity $O(N)$.

A linear array storing edges spanning any two consecutive layers is needed to preserve active edges in the scanning stack. The size of the array is limited by the height h of layers. Although in the worst case for the driving spanning tree with the shape of a linear sequence the number of arrays might reach the number of nodes in the entire network, practical cases operate with about $b \cdot h$ arrays, where b is the maximum branching factor (degree) of nodes and h is the height of layers. In real cases, both the branching factor and the height of layers are often limited. The processing of each node consists of the summation of at most b arrays. Therefore, if the order of the network is independent of the branching factor and the number of layers, then the number of possible edge crossings for each subtree can be computed in linear time $O(N)$.

4 IMPLEMENTATION, EXPERIMENTS, AND DISCUSSION

The algorithm was implemented as an experimental non-optimized Python script with some additional procedures evaluating the design process. It was used on an ultrabook DELL XPS 13 with 16GB of RAM using an Intel i7 2.7GHz processor. The layout and ordering are very fast, taking from seconds to hours for networks with a million nodes if all steps of the layout algorithms are performed in linear time. If the variant for the number of edge crossings with the quadratic asymptotic complexity is used, then the script can only be used for networks with up to 10^5 nodes. The linear variant for the number of edge crossings lasts roughly the same amount of time as the other steps of the layout algorithm. The processing of a large number of blocks, if the network contains blocks, during the undirected spanning tree selection remains the most difficult part of the processing chain.

We selected 20 datasets to evaluate the proposed methods (Pruitt, 2017; Leskovec and Krevl, 2017; GoogleFFT, 2017). Example datasets and their network statistics are shown in Table 1. Some datasets represent genealogical networks; the ITIS dataset is a snapshot of the Catalog of Life in the GEDCOM format (ITIS, 2017).

We provide additional dataset network properties related to edge crossings computation in Table 2. There were only two cases in which the processing needed to address strongly connected components. Stobie's dataset is the only one that contains 2 small strongly connected components. A deviation of a given network from a tree-like graph is characterized by the number of blocks with an order higher than 2. Also, the simplex based method of edge selection in blocks was rarely used (the column $|\mathcal{B}_S|$) because blocks often only form a single undirected cycle that can be solved directly without the ILP.

The improvement produced by the different criterion of subtree ordering based on the actual number of edge crossings can be observed in Table 2. The columns cr_T^o and cr_T^e represent the total numbers of edge crossings based on the original criterion using the graph order as the edge crossings estimate and the newly proposed criterion using the actual edge crossings spanning two consecutive layers, respectively. The improvement can be from 1% to 50% depending on the data form. If a given network is close to a tree, such as the ITIS network representing an overview of taxonomic information on plants, animals, fungi, and microbes as developed in the Integrated Taxonomic Information System (ITIS), then the improvement is

Table 1: Sample datasets and their statistics: a node number $|V|$ of the complete network , people number $|V_P|$ of the complete network, marriage number $|V_M|$ of the complete network, node number $|V_{max}|$ of the maximum component, edge number $|E_{max}|$ of the maximum component, number of layers $|L|$, number of source nodes $|V_{src}|$.

Dataset	$\|V\|$	$\|V_P\|$	$\|V_M\|$	$\|V_{max}\|$	$\|E_{max}\|$	$\|L\|$	$\|V_{src}\|$
Mykiska's network	2952	2192	765	2913	2917	27	609
USA presidents	3186	2145	1042	1589	1602	73	480
WeMightBeKin	52783	38486	14297	52672	54210	46	12716
ITIS	945352	472676	65799	615342	615341	36	1
Stobie's network	996055	706794	289268	995522	1038192	225	218593
FamiLinx	96693037	86124644	10568393	2276199	2480988	293	269637

Table 2: Layout processing statistics: number of strongly connected components $|SCC|$ of a size larger than 1, number of blocks $|\mathcal{B}|$, number of blocks with more than 2 nodes $||B_i| > 2|$, number of blocks processed by the simplex method $|\mathcal{B}_S|$, number of back edges $|E_{back}|$, number of nodes $|L_{\max}|$ in the maximum layer, total number of edge crossings cr_T^o if subtree order is used, total number of edge crossings cr_T^e if subtree edges spanning pairs of layers are used.

Dataset	$\|SCC\|$	$\|\mathcal{B}\|$	$\|\|B_i\| > 2\|$	$\|\mathcal{B}_S\|$	$\|E_{back}\|$	$\|L_{\max}\|$	$\|cr_T^o\|$	$\|cr_T^e\|$
Mykiska's network	0	2848	4	1	5	317	760	658
USA presidents	0	1132	3	1	14	97	223	110
WeMightBeKin	0	43848	19	2	1539	3374	16665	16283
ITIS	0	615341	0	0	0	58407	72	72
Stobie's network	2	768708	733	12	42671	42695	367278	355152
FamiLinx	0	884240	2125	188	204790	88299	2375287459	1584691327

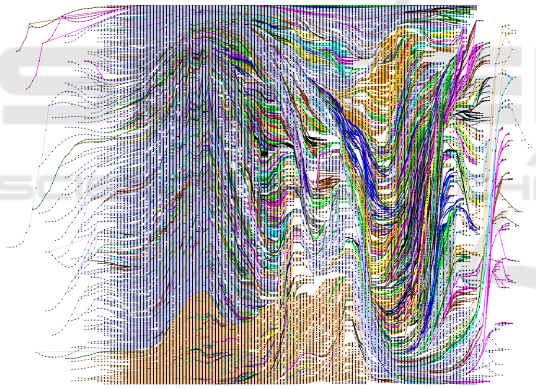

Figure 2: Thomas Stobie's network with almost one million nodes (people and their marriages). Each node is represented by a small dot. Ancestors are on the left, descendants on the right. Family clans (larger multitrees) are colored.

almost negligible. However, if the network has a multitree form with a minimum number of blocks with a size higher than 2, i.e. with minimum objects inheriting from other objects multiple times, then the number of edge crossings can be reduced significantly.

The volume performance of the method can also be demonstrated using the Stobie family network (Stobie, 2017) consisting of 995,522 nodes that is depicted in Fig. 2. Larger subtrees were highlighted using different colors. A node joining two such subtrees inherits the color of the larger subtree. Thus, one can observe flows of inheritance clans as colored lines. One such clan flow starts in the top left part of the diagram, it continues downward to the right bottom corner and then it creates a contemporary generation of people on the right side. Thus, from Fig. 2 it is easy to recognize that the network can be divided into two tree-like halves separated by this flow, one multitree covers the upper-right part with some colored lines visible, while the other multitree covers the bottom-left segment. A similar case study was applied to a society of the Old Kingdom of Egypt and an influence of spreading nepotism (inheritance of administration offices) was evaluated.

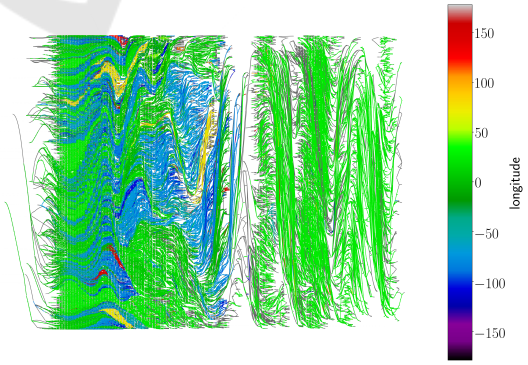

Figure 3: A fragment of the largest known family multitree (FamiLinx, 2018). The edges are colored according to the known locations (longitude) of people. If the location of a person is not available, then the color of the closest node is used.

We applied the layout method to a fragment of the largest known family multi-tree with over 13,588,042

nodes. This is the largest connected component in the dataset with 86,124,644 people and 51,807,142 edges (FamiLinx, 2018). A fragment with only 2,276,199 nodes was selected because it is currently the largest network that can be processed with the Python script in 16 GB of RAM. We removed all nodes with a degree 1 and selected branches of the largest component. The dataset was parsed in 123 minutes, the selection of the fragment ran for 108 minutes, strongly connected components were checked in 19 seconds, the driving spanning tree defining the node layers was computed in 208 seconds (including the processing of 2,125 blocks having more than 2 nodes and with one block having the order 1,364,449), the node order within layers was designed in 5 minutes, the position of nodes were calculated in 14 minutes and the picture was rendered in 3 hours. The nodes and edges are colored according to the locations of different people. Thus, one can observe how generations of people migrated across the Earth.

We do not compare our results to other methods as they use different layout criteria and different types of much smaller networks, often up to only 10,000 nodes. In fact, it would be unfair to compare methods that do impose node ordering or node grouping (e.g. siblings) with those which do not set such constraints, or methods that can process general acyclic graphs using higher complexity techniques with our restricted approach only focused on multitree structures. We are not aware of any implementation that tries to solve a problem similar to ours.

5 CONCLUSION

In this work, we proposed the modification of criterion controlling subtree ordering during multitree-like network layout design using driving spanning tree. The proposed criterion is based on the actual number of edge crossings injected by a given subtree ordering. It was shown that the possible number of edge crossing injected by any subtree of the driving spanning tree can be computed efficiently in linear time for practical cases. Thus, the complete layout design could be performed in almost linear time. The optimum spanning subtree selections based on the processing of blocks, although they are very rare, remains the most critical step in spanning tree selection. The new proposed feasible criterion decreases the number of edge crossings from 1% to 50% as the network departs from the pure tree form towards multitree-like forms.

The method is very efficient for layered multitree-like network layouts with constraints on node order concering their layers and their order in layers. The produced graph layouts are more acceptable for the user if they deal with large networks combining many trees into a single acyclic graph. As the driving spanning tree and all other processing steps can be computed very efficiently for multitree-like networks, it is possible to process networks with millions of nodes. The current memory based unoptimized implementation written in Python limits the use of the proposed method to networks with up to 4 million nodes on a computer with 16GB of RAM. Such network layouts contribute significantly to the comprehension of vast networks and their basic structural top-level patterns, e.g. this enables making decisions on their processing. The implementation of a special tool allowing panning and zooming above such layouts is beyond the scope of this paper.

ACKNOWLEDGEMENTS

Sponsored by the project for GAČR, No. 16-072105: Complex network methods applied to ancient Egyptian data in the Old Kingdom (2700–2180 BC).

REFERENCES

Bondy, J. and Murty, U. (2008). *Graph Theory*. Springer.

Booth, K. S. and Lueker, G. S. (1976). Testing for the consecutive ones property, interval graphs and graph planarity using PQ-tree algorithms. *Journal of Computer and System Sciences*, 13(3):335–379.

Brandes, U. and Köpf, B. (2002). *Graph Drawing: 9th International Symposium, GD 2001 Vienna, Austria, September 23–26, 2001 Revised Papers*, chapter Fast and Simple Horizontal Coordinate Assignment, pages 31–44. Springer Berlin Heidelberg, Berlin, Heidelberg.

Chimani, M., Gutwenger, C., Mutzel, P., and Wong, H.-M. (2011). Upward planarization layout. *Journal of Graph Algorithms and Applications*, 15(1):127–155.

Chimani, M., Junger, M., and Schulz, M. (2008). Crossing minimization meets simultaneous drawing. In *2008 IEEE Pacific Visualization Symposium*, pages 33–40.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009). *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition.

Diestel, R. (2005). *Graph Theory*. Springer.

Eades, P., McKay, B. D., and Wormald, N. C. (1986). On an edge crossing problem. In *Proc. ACSC'86, Australian National University*, pages 327–334.

FamiLinx (2018). Familinx dataset, accessed 2018-12-01. http://familinx.org/data.html.

Forster, M. (2005). A fast and simple heuristic for constrained two-level crossing reduction. In *Graph Drawing. GD 2004. Lecture Notes in Computer Science, vol 3383. Springer, Berlin, Heidelberg*, pages 206–216.

Gansner, E. R., Koutsofios, E., North, S. C., and Vo, K. P. (1993). A technique for drawing directed graphs. *IEEE Transactions nn Software Engineering*, 19(3):214–230.

Gansner, E. R. and North, S. C. (2000). An open graph visualization system and its applications to software engineering. *Softw. Pract. Exper.*, 30(11):1203–1233.

Gibson, H., Faith, J., and Vickers, P. (2013). A survey of two-dimensional graph layout techniques for information visualisation. *Information Visualization*, 12(3-4):324–357.

GoogleFFT (2017). Google: Famous family trees. https://groups.google.com/forum/#forum/famous-family-trees.

Graphviz (2016). Graphviz - graph visualization software. www.graphviz.org. Accessed: 5.6.2016.

Healy, P. and Nikolov, N. S. (2003). Characterization of layered graphs with the minimum number of dummy vertices. Technical Report UL-CSIS-03-4, CSIS Department, University of Limerick, Limerick, Republic of Ireland.

Healy, P. and Nikolov, N. S. (2013). *Handbook of Graph Drawing and Visualization*, chapter Hierarchical Drawing Algorithms, pages 409–453. CRC Press.

Hopcroft, J. and Tarjan, R. (1973). Algorithm 447: Efficient algorithms for graph manipulation. *Commun. ACM*, 16(6):372–378.

Hopcroft, J. and Tarjan, R. (1974). Efficient planarity testing. *Journal of the ACM*, 21(4):549–568.

Hsu, W.-L. and McConnell, R. (2004). *Handbook of Data Structures and Applications*, chapter PQ Trees, PC Trees, and Planar Graphs, pages 32–1–32–27. CRC Press.

ITIS (2017). ITIS - Integrated Taxonomic Information System. https://www.itis.gov/downloads/index.html. Retrieved February, 10, 2017, from the Integrated Taxonomic Information System on-line database, http://www.itis.gov.

Lempel, A., Even, S., and Cederbaum, I. (1967). An algorithm for planarity testing of graphs. In Rosenstiehl, P., Gordon, and Breach, editors, *Theory of Graphs*, pages 215–232, New York.

Leskovec, J. and Krevl, A. (2017). SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data.

Lutteropp, S. (2014). On layered drawings of planar graphs. Master's thesis, Karlruhe Institute of Technology.

Marik, R. (2016). Tree-based genealogical graph layout. In Hu, Y. and Nöllenburg, M., editors, *Graph Drawing and Network Visualization, 24th International Symposium, GD 2016, Athens, Greece, September 19-21*, volume ISBN: 978-3-319-50105-5 (Print) 978-3-319-50106-2 (Online).

Marik, R. (2017a). *Efficient Genealogical Graph Layout*, pages 567–578. Springer International Publishing, Cham.

Marik, R. (2017b). *On Multitree-Like Graph Layering*, pages 595–606. Springer International Publishing, Cham.

Marik, R. (2018). Layered graph force-driven vertex positioning. In *Proceedings of the 13th International In Proceedings of the 13th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP 2018) Funchal, Madeira, Portugal, 27-29 January, IVAPP 2018*, volume 3: IVAPP, pages 301–308.

Mathews, E. and Frey, H. (2012). *Distributed Computing and Networking: 13th International Conference, ICDCN 2012, Hong Kong, China, January 3-6, 2012. Proceedings*, chapter A Localized Link Removal and Addition Based Planarization Algorithm, pages 337–350. Springer Berlin Heidelberg, Berlin, Heidelberg.

Nikolov, N. S., Tarassov, A., and Branke, J. (2005). In search for efficient heuristics for minimum-width graph layering with consideration of dummy nodes. *J. Exp. Algorithmics*, 10.

Paton, K. (1971). An algorithm for the blocks and cutnodes of a graph. *Commun. ACM*, 14(7):468–475.

Pruitt, P. D. (2017). Great sites for links to genealogy software. http://famousfamilytrees.blogspot.cz/2011/12/. Accessed: February 2017.

Reingold, E. M. and Tilford, J. S. (1981). Tidier drawings of trees. *IEEE Transactions on Software Engineering*, SE-7(2):223–228.

Resende, M. G. C. and Ribeiro, C. C. (2001). *Encyclopedia of Optimization*, chapter Graph planarization, pages 908–913. Springer US, Boston, MA.

Shih, W.-K. and Hsu, W.-L. (1999). A new planarity test. *Theoretical Computer Science*, 223(1-2):179–191.

Stobie, T. (2017). Thomas stobie's genealogy pages. http://freepages.genealogy.rootsweb.ancestry.com/ stobie/. Accessed: February 2017.

Sugiyama, K. and Misue, K. (1991). Visualization of structural information: automatic drawing of compound digraphs. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(4):876–892.

Sugiyama, K., Tagawa, S., and Toda, M. (1981). Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(2):109–125.

Tutte, W. T. (1963). How to draw a graph. *Proceedings of the London Mathematical Society, Third Series*, 3(13):743–768.

Warfield, J. N. (1977). Crossing theory and hierarchy mapping. *IEEE Transactions on Systems, Man, and Cybernetics*, 7(7):505–523.

Wilson, R. J. (1998). *Introduction to Graph Theory*. Longman, fourth edition.