Efficient Genealogical Graph Layout

Radek Marik

Abstract While a visual unconstrained tree structure planar layout design is easy to implement, a visualization of a tree with constraints on node ranks and their ordering within ranks leads to a difficult combinatorial problem. A genealogical graph, such as family tree, can be taken as an example of such a case. Classical ancestor trees, descendant trees, Hourglass charts, and their visual variants such as node-link diagrams or fan charts are suitable for assessment of peoples relationships when one is focused on a particular person and his/her direct ancestors and descendants. Such tree-based representations miss a broader context of relationships and do not allow the quick assessment of several interlinked families together. We propose a new undirected tree-driven layout technique for layered multitree graph visualizations producing constraints on node layers and ordering of groups of nodes within layers. The computed constraints can be mapped, at least partially, into the DOT language property directives used by the Graphviz toolbox. We demonstrate achievements on several datasets containing up to 39000 people.

1 Introduction and Related Methods

Although it is more than 55 years since Tutte introduced barycentric embedding, research of graph visualization techniques remains a highly active field attracting a lot of attention [16, 34, 35]. Graph visualization can help to form an overview of relational patterns and detect data structure much faster than data in a tabular form. The form in which the graph is presented has a significant impact on how the graph is understood and the time that is necessary to achieve this. Nodes placed close to one another might be interpreted by the user as a true relationship whether or not this

Radek Marik (e-mail: `Radek.Marik@fel.cvut.cz`)✉
Department of Telecommunication Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague, Technicka 2, Dejvice, Prague, CZ-166 27, Czech Republic
WWW home page: `https://comtel.fel.cvut.cz/en/users/marikr`

© Springer International Publishing AG 2017

H. Cherifi et al. (eds.), *Complex Networks & Their Applications V*,
Studies in Computational Intelligence 693,
DOI 10.1007/978-3-319-50901-3_45

relationship exists [16, 26]. Working with genealogical graphs is no exception in this sense.

Tree based drawing methods of genealogical graphs have been among the standard techniques for centuries. Ancestor trees, descendant trees and Hourglass charts belong to a set of traditional tools [21] implemented by a majority of freeware, shareware, or commercial tools, for example Gramps [1] or MyHeritage [3]. These tools provide a clear description of a situation when the user needs to investigate direct ancestors and/or descendants of a given person (often referred to as the main or center person) placed into the root of the tree. Thus, the generation of the main person consists of only one person and the size of other generations grows exponentially with a branching factor often over 2. Therefore, the classical node-link tree graphical representation resulting in a triangular shape wastes about one half of the drawing area. There are other more space-efficient representations such as fan charts or H-charts [5, 22, 36, 38]. As any pure tree representation enables any ordering of node predecessors/successors, it is possible to specify the type of ordering, such as children ordered by their birth dates. It is also possible to extend any such tree representation with additional nodes that can be attached as single nodes to any tree node (in the Gramps tool [1] this type of graph is called a Relationship Graph). In this way a tree with direct ancestors/descendants can cover, for example, spouses/partners. Therefore, tree representations can be laid out in such a way that family members are grouped together. The obvious drawback of the pure tree representations is that selecting a different main person leads to a different graph that must be rendered again. Such tree-based representations also miss a broader context of relationships and do not allow the quick assessment of several interlinked families together.

However, the situation with family member grouping changes significantly if the assumptions of one main person and direct ancestors/descendants are dropped. In a number of cases it is highly beneficial if the entire network of families, or at least a significant part, can be displayed in one layout. Then we face issues with challenges linked to edge crossing and preferences on node clustering [32, 33, 37]. Therefore, the standard techniques for planar graph layouts [6, 19, 20, 23, 29, 31] including planarization techniques [8, 9, 25, 30] are not suitable in all cases. Methods designed for layered graphs aimed at exact solutions [37] do not scale to large graphs; methods related to two-layer crossing problems using either averaging heuristics, such as the barycenter and the median methods [17, 33], or a hybrid approach [13, 15], produce layouts far from the optimum; and a local node order propagation [24] cannot resolve more global node order constraints.

The genealogical tools often use methods proposed for a general graph layout, such as hierarchical layouts, for example, implemented and provided by tools such as `dot.exe` (DOT) in Graphviz package [2] or yEd [4]. Unfortunately, these tools, and others we are aware of, do not support any kind of constraints that would allow the setting of node cluster preferences.

Assuming that a genealogical graph is layered according to the generation levels determined by an algorithm, such as the one proposed in the next section, the main complaint stems from mixing of children/partners from different families. Based on our own experience and observations made during our cooperation with

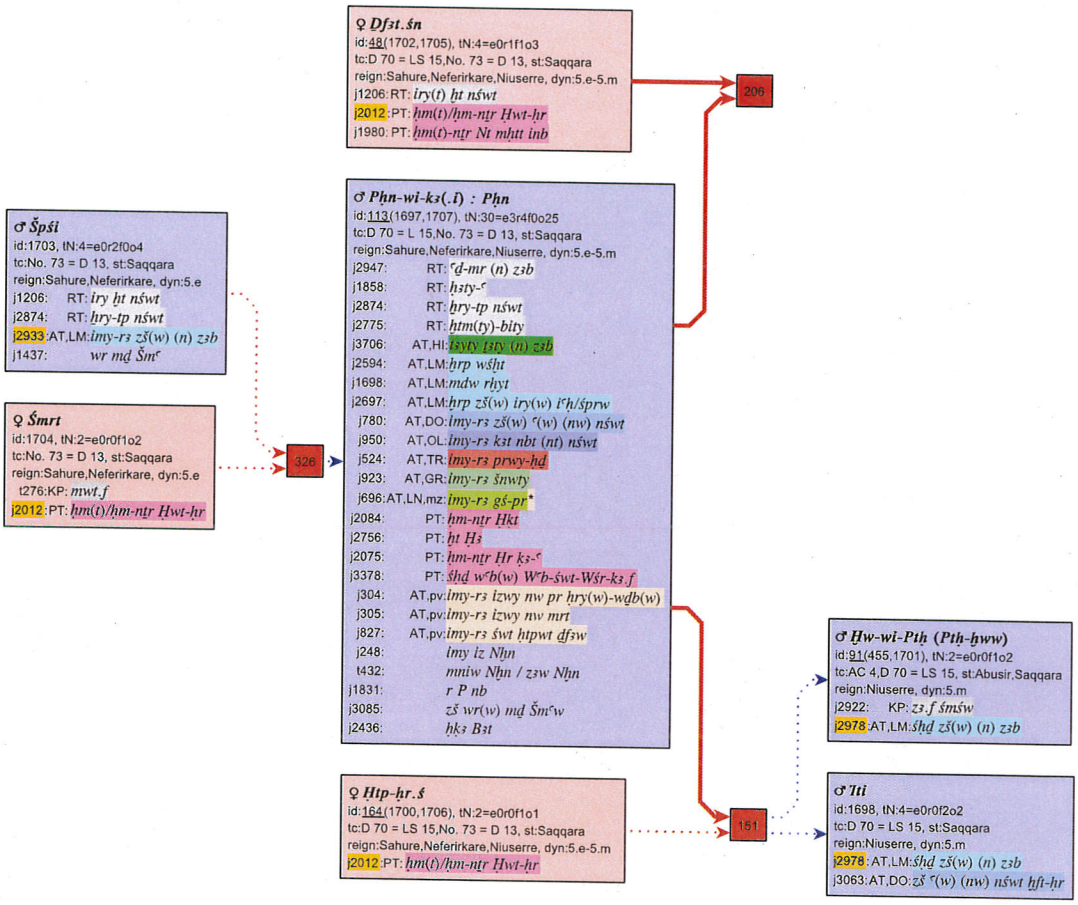

Fig. 1: A family tree component presented using a tree layout which is illustrative of Vizier Pehenuikas family. The people rectangles contain information such as their titles and offices.

Egyptologists, the researchers prefer grouping based on families. For example, Fig. 1 depicts Vizier Pehenuikas family reconstructed from the database of the Egyptian officials [12]. In this case, the layout was produced using the yEd tool.

When several families linked through a partnership relationship are visualized, one can cluster either children or partners, but generally not both. For example, Relationship graph visualization implemented in the DOT creates subgraphs of partners. Unfortunately, directed hierarchical drawing methods such as the very good one implemented as `dot.exe` [15] results in layouts mixing generations and members of several families, see Fig. 2. Children are often ordered in families randomly, furthermore children might be assigned to different ranks, children from different families might be interleaved, and a number of edge crossings occur. Such layouts are difficult to read and comprehend.

Graph specifications do not contain usually any constraints on node layers. Layer layout implementations rank nodes as proposed by many authors [15, 33]. In many situations the resulting layout is produced as required. Unfortunately, general criteria lead to node placement breaking generation layering as is usual and expected in genealogical graphs, i.e. children of one family at the same level and similarly their parents, see Fig 2. Some implementations, such as the DOT language, enable a

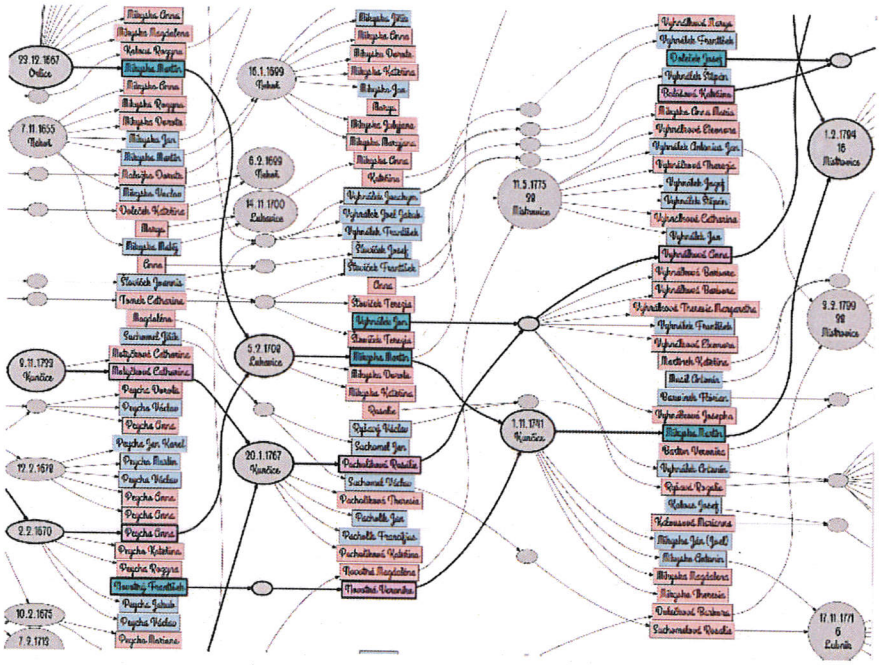

Fig. 2: A sample partial snapshot of a private family tree consisting of 2192 people as rendered using the DOT tool without any further constraints. Colored rectangles represent people (reddish for women, blueish for men). Ovals capture their marriages. Although the visualization seems to be correct, there are many cases when people are moved into different generation layers, many children from different families are mixed, and a number of edge crossings can be avoided.

specification that a subset of nodes shares the same layer (rank). The majority of algorithms computing ranks are derived from the topological order computation ($O(n)$ time complexity) [10] and select one of many possible solutions that satisfy layer intervals of node placements. Classical algorithms start from a single node, the only one with no predecessors. Generally, a genealogical graph can consist of several nodes without predecessors and several nodes without successors.

Formally, a genealogical graph is an acyclic bipartite directed graph $G(V_P, V_M, E)$ with two sorts of nodes, people V_P and marriages/partnerships V_M. The edges E are directed from parent nodes to marriage nodes and from marriage nodes to children nodes. If a family tree contains multiple marriages from one family to another, but it does not contain marriages between any two blood relatives, then it forms a multitree [14, 27]. A layering of an acyclic digraph $G(V_P, V_M, E)$ is a partition of $V_P \cup V_M$ into subsets $L_1, L_2, \ldots, L_h$, such that if $(u, v) \in E$, where $u \in L_i$ and $v \in L_j$, then $i < j$ [7]. Without loss of generality we can assume that the index of the generation layer of parents (also denoted as ranks) is lower than the index of their marriage node, and further that the index of the marriage node is lower than the index of children nodes. In this paper, a (total) node order reflects a linear sequence of nodes in a given layer.

We are not aware of any method that would enable the definition and use of the necessary layout constraints. Recently, it was demonstrated using two simple propagated node order constraints that node layouts of such graphs can be improved

significantly [24]. The proposed topological layout technique in [24] is based on a local propagation of children and parent ordering across generations (ranks), but it is not able to reflect global subtree constraints.

In this paper we propose a new method that allows the determination of such node order constraints using an undirected tree-driven layout of subtrees and does not exhibit deficiencies of the local propagation [24]. The new method produces significantly fewer edge crossings than the methods mentioned above. At least partially, the proposed constraints can be mapped to additional graph specifications that result in the DOT algorithm producing the required layout. More specifically, our method modifies the first two steps of the approach proposed in [15, 33], i.e. 1/ determination of layers (generations, node ranks, levels), and 2/ enforcing node orders within the layers. In provided visualizations, layers define a horizontal index while orders are reflected by a vertical index.

The rest of the paper is organized in the following way. In the next section we present an algorithm that allows setting layers of nodes for an acyclic graph representing a traditional representation of family tree using marriage nodes. Then we describe the technical details of the new proposed method calculating node orders within the layers. Finally, we discuss achieved results and tests on real data datasets with thousands of nodes.

2 Layering of Genealogical Graph Nodes

In this section we present an algorithm already proposed in [24], using which the ranks of nodes can be determined for any genealogical graph. In the algorithm we assume that the processed graph is directed and acyclic. Let us use a convention that node ranks are identified by numbers $\lambda_2(v)$ and successors have higher levels. Each node is assigned an interval of rank levels at which the node can appear with regard to a base level. The algorithm uses two simple passes through a graph. Each node is assigned the highest possible level with respect to the current highest base level of successors during the first pass. In fact, this pass assigns node layers conforming to longest path layering [17].

$$\lambda_1(v) = \begin{cases} \max_{(v,w)\in E} \lambda_1(w) - 1 & \text{if } v \text{ has successors} \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

Thus, the node(s) with the lowest level can be determined. A rank level for each node is set as the maximum level of the node predecessor levels increased by one during the second pass. The second pass starts from the nodes with the lowest level.

$$\lambda_2(v) = \begin{cases} 0 & \text{if } v \text{ has the lowest level} \\ \lambda_2(w) - 1 & \text{if } w \text{ has predecessors partially processed} \\ & (v,w) \in E \text{ and } \lambda_2(v) \text{ is not assigned} \\ \min_{(w,v)\in E} \lambda_2(w) + 1 & \text{if } v \text{ has all predecessors processed} \end{cases} \tag{2}$$

Each node is visited twice during each pass using depth first search (DFS) using an explicit LIFO queue. The first visit ensures that all successors/predecessors are processed already. When the node is visited again, its level is determined as minimum/maximum of successors/predecessors levels. As children from a single marriage have only one common predecessor, the marriage node, they share the same rank level. However, parent nodes can be assigned to different levels. Nevertheless, the algorithm guarantees that parents linked to a marriage node always have a lower layer number than the marriage node and children attached to the marriage node have higher layer numbers than the marriage node. The algorithm uses two DFS passes with linear time complexity $O(N)$, where N is the number of graph nodes.

3 Nodes Ordering within Layers

In this section we support an approach that results in siblings of one family being clustered tightly while partnerships/parents might be mixed. The obvious reason behind this variant is that the number of children is much higher than 2, often reaching values over 10. Thus, an injected edge crossing because of mixed parents is much lower than occurs when children are mixed, and families can be identified easily by a number of parallel edges leading from marriage nodes to children nodes.

The problem of a layout design might then be reduced to a determination of the order of people belonging to one generation layer. We propose that children belonging to a single family are ordered by their birth dates. Subtrees of the child descendants, including descendant marriage nodes, hold this order. In the opposite direction, i.e. from a marriage node to its spouses, the order of spouses can be determined according to birthdates of spouses. There might be cases when two or more people from two or more different families create partnerships. In such situations we cannot insist on the order of marriage nodes as the order requirements might be contradictory, for example, in the case of two families both with two children that creates two marriages in the opposite order of their birthdates.

As cases when two individuals share two or more distinct subtrees are very rare in reality (just 5 cases in our database of 2192 individuals), we can transform the genealogical graph into an undirected tree by removing a few edges. The undirected tree can be decomposed into subtrees. These subtrees are layered recursively as strips side by side while following a simple set of rules that minimize edge crossings. Nodes of processed subtrees are placed into rank arrays during the subtree layering. Thus, the rank array determines the order of their nodes. The whole method can be described using four steps:

1. Node rank determination
2. Undirected spanning tree subgraph selection
3. Computation of shape characteristics for all subtrees
4. Node order design by subtrees layering.

The used node rank determination was already described in Sect. 2. Thus, we are going to focus on the other three steps. Structures driving node order design are created in steps 2 and 3. The actual layout is performed in the last step.

3.1 Undirected spanning tree subgraph selection

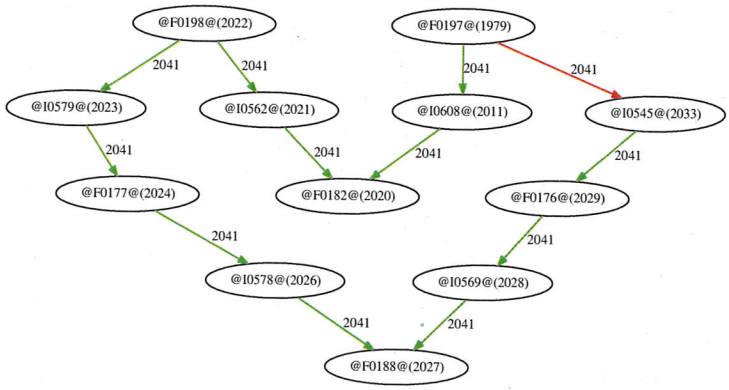

Fig. 3: A detected block of the sample family tree database. The backedge is red. One can deduce from the block that descendants of the marriage node `@F0188@` share ancestors of two distinct subtrees `@F0198@` and `@F0197@`.

The layout design is controlled by an undirected spanning tree of the original directed graph. Removing backedges using a DFS scan is a sufficient simple solution. These removed edges are drawn but not used by the node ordering algorithm. However, we might select other edges to stress their particular role in the graph because the removed edges typically cause edge crossings. We select suitable edges using blocks (biconnected components [11]) in linear time [18, 28] to break cycles (a DFS scan, cycles determined by backedges), see Fig. 3. Blocks might occur if two or more individuals share two or more distinct ancestor subtrees. A related analysis of the blocks is considered beyond the scope of this paper.

3.2 Subtree shape characteristics

As we will see later, the graph is layered starting from a node with the lowest rank. The layout technique makes decisions based on subtree orders (the number of nodes) and comparisons of the current node rank with the minimum node ranks of processed subtrees. Starting from the node with the lowest rank we assign both pre/post order timestamps τ_1, τ_2 to each node using the DFS scanning [10]. The order of any undirected subtree can be calculated as $(\tau_2^r - \tau_1^r)/2$, where τ_1^r, τ_2^r are the timestamps of the subtree root, because each node of the subtree has just two timestamps.

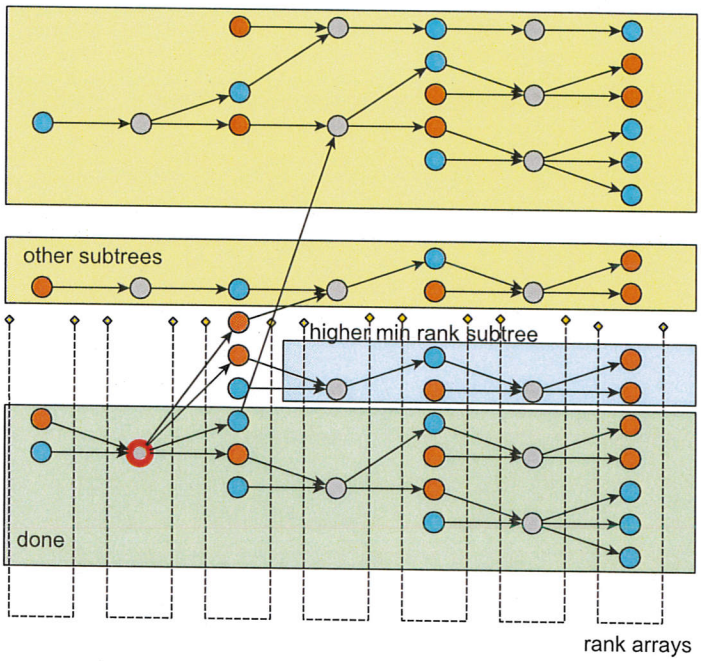

Fig. 4 A symbolical snapshot of the proposed layout method. The current node is tagged by its thick red border. Blue nodes represent men, orange nodes represent women, gray nodes represent their marriages. The greenish zone is an already processed part of the graph with all nodes registered in the rank arrays that keep their order of registration. The blue zone contains just one subtree with the minimum rank higher than the rank of the current node. Two yellow zones represent another two subtrees in the order in which they be layered based on their number of nodes.

Using the post order we can determine the minimum node rank $\Lambda(v_i)$ of any subtree determined by its root node v_i inside its timestamp interval $[\tau_1(v_i), \tau_2(v_i)]$ in $O(N)$. The $NB(v_i)$ function produces undirected neighbor nodes of v_i.

$$\Lambda(v_i) = \min(\lambda_2(v_i), \min_{v_j \in NB(v_i):\tau_2(v_j)<\tau_2(v_i)} \Lambda(v_j)) \tag{3}$$

3.3 Design of node order within layers

Again starting from the node with the lowest rank we assign nodes of timestamp interval subtrees into rank arrays (initially empty for each rank), see Fig. 4. First, subtrees with a minimum rank higher than the rank 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 size to minimize edge crossing, because it is expected that an edge linking the current node to a subtree and crossing other larger graph subtrees produces more edge crossings. Let us assume we process the current node where some children nodes link K subtrees with a minimum rank lower than the current node rank. A sequence $[cr_1, \ldots cr_K]$ is obtained if the subtrees are sorted according to their edge crossing counts cr_ℓ between the children 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 minimum if the sequence $[cr_1, \ldots cr_K]$ is not decreasing.

The method can be described as the following sequence of steps:

1. Set an empty array for each node rank.
2. Set an empty LIFO stack of processed nodes.
3. Add the node with the lowest rank to the stack.

4. Pop a node v_i from the stack.
5. Register the node to the appropriate rank array given by $\lambda_2(v_i)$.
6. Register all children of v_i and their spouses with empty ancestor subtrees in the required order given by birthdates.
7. Select nodes $v_j \in NB(v_i)$ with $\Lambda(v_j) <= \lambda_2(v_i)$ and add them to the stack sorted by the decreasing size of their subtrees given by $\tau_2(v_j) - \tau_1(v_j)$.
8. Select nodes $v_j \in NB(v_i)$ with $\Lambda(v_j) > \lambda_2(v_i)$ and add them to the stack sorted by birthdates.
9. If the stack is empty, then stop, otherwise continue with step 4.

The algorithm is a kind of DFS scanning with the linear complexity $O(N)$ at the top level. The selections of nodes in steps 7 and 8 can be performed in linear time, too. Steps 7 and 8 also performs sorting with complexity $O(N_v \log(N_v))$ where N_v is a branching factor which is a very low number in genealogical graphs. N_v is often limited by value 15 (a maximum number of children and two parents) so the complexity of steps 7 and 8 can be treated as a constant. Thus, the overall complexity of the layout method is close to $O(N)$.

4 Implementation, Experiments, and Discussion

We implemented the proposed first two steps of an acyclic genealogical graph layout algorithm. The steps produce the constraints on generation layers and node orders in each generation.

We selected 20 datasets for an evaluation of the proposed constraints contribution. The first dataset consists of 2192 people of the authors private family relationship genealogical graph. The set is created as a merge of several family trees ranging over 14 generations with the first records dated 1647. The second dataset consists of 3057 people of the database created by Egyptologists [12]. The database covers high ranking officials from the 4th, 5th, and 6th dynasties and their families. One can reconstruct over 160 families with up to 6 generations. The database has been filled over ten years. Generated graphs covering more families greatly help Egyptologists to assess quickly investigated social phenomena. Experiments with families of the Egyptian database did not exhibit any layout deficiencies as the families are quite simple and not larger than 50 family members. Similar results were obtained for the rest of the datasets that are GEDCOM files downloaded from the Internet with from 400 to 39,000 individuals.

Layout constraints generated using the method proposed in this paper are depicted in Fig 5. The layers of nodes were placed uniformly in the horizontal direction while their ordered nodes were placed uniformly in the vertical direction. The nodes were linked with straight-lined edges. Family clans are kept well separated. The layout is created very quickly (below 0.5 second with a Python script on DELL XPS 13 using an Intel i7 2GHz processor.

The method can be further improved. We have not considered the correct number of edge crossings, but only its estimate. The layout design adds subtree layers only on one side. It could utilize both sides of the current node. In fact, a better method could

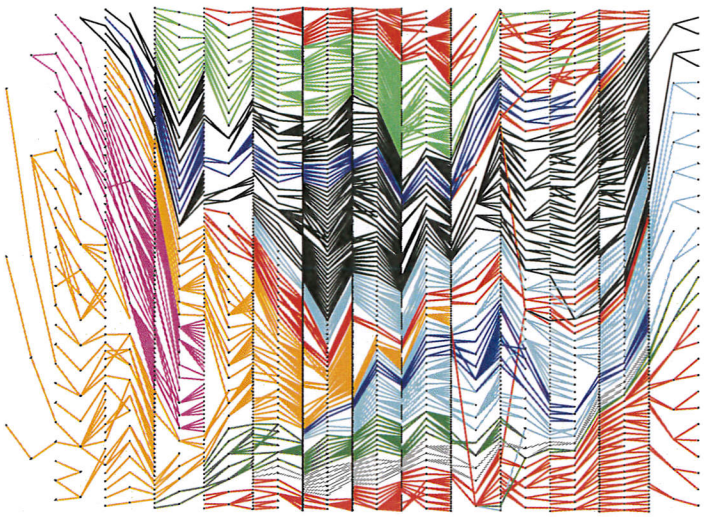

Fig. 5: A very low edge crossing visualization of the sample private family tree with 2192 individuals and 765 marriages created by the proposed method. Family clans with more than 150 people are emphasized with different colors. An ideal layout would result in edges creating *waves* only.

utilize a combinatorial assessment of subtree minimum ranks combined with subtree width at the current node rank. However, the complexity of such a method would be much higher while the gain in edge crossings would be minimal (considering the current datasets characteristics).

5 Conclusion

In this work we proposed a new method for a ranked multitree layout with constraints on node order and its layers. In fact, the constraints result in a fully specified topological arrangement of the graph nodes in plane. The constraints can be computed very efficiently. The experiments demonstrate clearly a significant improvement in graph comprehension because of low edge crossing and compact family grouping and indicate that the results provided by the present state of the art tools are quite far from the optimum layout, at least for special sorts of graphs such as genealogical ones. The smaller number of undirected backedges the better layout results.

Acknowledgements Sponsored by the project for GAČR, No. 16-072105: Complex network methods applied to ancient Egypt data in the Old Kingdom (27002180 BC).

References

[1] Gramps. genealogical research software. `https://gramps-project.org/` (2016). Accessed: 5.6.2016
[2] Graphviz - graph visualization software. `www.graphviz.org` (2016). Accessed: 5.6.2016
[3] Myheritage. `https://www.myheritage.cz` (2016). Accessed: 5.6.2016

[4] yed graph editor. http://www.yworks.com/products/yed (2016). Accessed: 5.6.2016

[5] Ball, R., Cook, D.: A family-centric genealogy visualization paradigm. In: 14th Annual Family History Technology Workshop. Provo, Utah (2014)

[6] Booth, K.S., Lueker, G.S.: 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 (1976)

[7] Brandes, U., Köpf, B.: Fast and Simple Horizontal Coordinate Assignment, pp. 31–44. Springer Berlin Heidelberg, Berlin, Heidelberg (2002). DOI 10.1007/3-540-45848-4_3. URL http://dx.doi.org/10.1007/3-540-45848-4_3

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

[9] Chimani, M., Junger, M., Schulz, M.: Crossing minimization meets simultaneous drawing. In: 2008 IEEE Pacific Visualization Symposium, pp. 33–40 (2008). DOI 10.1109/PACIFICVIS.2008.4475456

[10] Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, Third Edition, 3rd edn. The MIT Press (2009)

[11] Diestel, R.: Graph Theory. Springer (2005)

[12] Dulíková, V.: The reign of king Nyuserre and its impact on the development of the Egyptian state. A multiplier effect period during the Old Kingdom. Ph.D. thesis, Charles University in Prague, Faculty of Arts, Czech Institute of Egyptology (2016)

[13] Eiglsperger, M., Siebenhaller, M., Kaufmann, M.: An Efficient Implementation of Sugiyama's Algorithm for Layered Graph Drawing, pp. 155–166. Springer Berlin Heidelberg, Berlin, Heidelberg (2005). DOI 10.1007/978-3-540-31843-9_17. URL http://dx.doi.org/10.1007/978-3-540-31843-9_17

[14] Furnas, G.W., Zacks, J.: Multitrees: Enriching and reusing hierarchical structure. In: Conference Companion on Human Factors in Computing Systems, CHI '94, pp. 223–. ACM, New York, NY, USA (1994). DOI 10.1145/259963.260396. URL http://doi.acm.org/10.1145/259963.260396

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

[16] Gibson, H., Faith, J., Vickers, P.: A survey of two-dimensional graph layout techniques for information visualisation. Information Visualization **12**(3-4), 324–357 (2013). DOI 10.1177/1473871612455749. URL http://ivi.sagepub.com/content/12/3-4/324.abstract

[17] Healy, P., Nikolov, N.S.: Handbook of Graph Drawing and Visualization, chap. Hierarchical Drawing Algorithms, pp. 409–454. CRC (2013)

[18] Hopcroft, J., Tarjan, R.: Algorithm 447: Efficient algorithms for graph manipulation. Commun. ACM **16**(6), 372–378 (1973). DOI 10.1145/362248.362272. URL http://doi.acm.org/10.1145/362248.362272

[19] Hopcroft, J., Tarjan, R.: Efficient planarity testing. Journal of the ACM **21**(4), 549–568 (1974). DOI 10.1145/321850.321852. URL http://doi.acm.org/10.1145/321850.321852

[20] Hsu, W.L., McConnell, R.: Handbook of Data Structures and Applications, chap. PQ Trees, PC Trees, and Planar Graphs, pp. 32–1–32–27. CRC Press (2004)

[21] Keller, K., Reddy, P., Sachdeva, S.: Family tree visualization. Course project report. http://vis.berkeley.edu/courses/cs294-10-sp10/wiki/images/f/f2/Family_Tree_Visualization_-_Final_Paper.pdf (2010). Accessed: 5.6.2016

[22] Kieffer, S., Dwyer, T., Marriott, K., Wybrow, M.: Hola: Human-like orthogonal network layout. IEEE Transactions on Visualization and Computer Graphics **22**(1), 349–358 (2016). DOI 10.1109/TVCG.2015.2467451

[23] Lempel, A., Even, S., Cederbaum, I.: An algorithm for planarity testing of graphs. In: P. Rosenstiehl, Gordon, Breach (eds.) Theory of Graphs, pp. 215–232. New York (1967)

[24] Marik, R.: On large genealogical graph layouts (accepted for publication). In: WASACNA 2016 : Workshop on Algorithmic and Structural Aspects of Complex Networks and Applications, September 17th, Tatransk Matliare, Slovakia (2016)

[25] Mathews, E., Frey, H.: A Localized Link Removal and Addition Based Planarization Algorithm, pp. 337–350. Springer Berlin Heidelberg, Berlin, Heidelberg (2012). DOI 10.1007/978-3-642-25959-3_25. URL http://dx.doi.org/10.1007/978-3-642-25959-3_25

[26] McGrath, C., Blythe, J., Krackhardt, D.: Seeing groups in graph layouts. Connections **19**(2), 22–29 (1996)

[27] McGuffin, M.J., Balakrishnan, R.: Interactive visualization of genealogical graphs. In: IEEE Symposium on Information Visualization, 2005. INFOVIS 2005., pp. 16–23 (2005). DOI 10.1109/INFVIS.2005.1532124

[28] Paton, K.: An algorithm for the blocks and cutnodes of a graph. Commun. ACM **14**(7), 468–475 (1971). DOI 10.1145/362619.362628. URL http://doi.acm.org/10.1145/362619.362628

[29] Reingold, E.M., Tilford, J.S.: Tidier drawings of trees. IEEE Transactions on Software Engineering **SE-7**(2), 223–228 (1981). DOI 10.1109/TSE.1981.234519

[30] Resende, M.G.C., Ribeiro, C.C.: Graph planarizationGraph Planarization, pp. 908–913. Springer US, Boston, MA (2001). DOI 10.1007/0-306-48332-7_187. URL http://dx.doi.org/10.1007/0-306-48332-7_187

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

[32] Sugiyama, K., Misue, K.: Visualization of structural information: automatic drawing of compound digraphs. IEEE Transactions on Systems, Man, and Cybernetics **21**(4), 876–892 (1991). DOI 10.1109/21.108304

[33] Sugiyama, K., Tagawa, S., Toda, M.: Methods for visual understanding of hierarchical system structures. IEEE Transactions on Systems, Man, and Cybernetics **11**(2), 109–125 (1981). DOI 10.1109/TSMC.1981.4308636

[34] Tutte, W.T.: Convex representations of graphs. Proceedings of the London Mathematical Society, Third Series (10), 304–320 (1960)

[35] Tutte, W.T.: How to draw a graph. Proceedings of the London Mathematical Society, Third Series (13), 743–768 (1960)

[36] Tuttle, C., Nonato, L.G., Silva, C.: Pedvis: A structured, space-efficient technique for pedigree visualization. IEEE Transactions on Visualization and Computer Graphics **16**(6), 1063–1072 (2010). DOI 10.1109/TVCG.2010.185

[37] Warfield, J.N.: Crossing theory and hierarchy mapping. IEEE Transactions on Systems, Man, and Cybernetics **7**(7), 505–523 (1977). DOI 10.1109/TSMC.1977.4309760

[38] Yoghourdjian, V., Dwyer, T., Gange, G., Kieffer, S., Klein, K., Marriott, K.: High-quality ultra-compact grid layout of grouped networks. IEEE Transactions on Visualization and Computer Graphics **22**(1), 339–348 (2016). DOI 10.1109/TVCG.2015.2467251