ITAT 2016 Proceedings, CEUR Workshop Proceedings Vol. 1649, pp. 218–225
http://ceur-ws.org/Vol-1649, Series ISSN 1613-0073, © 2016 R. Mařík

# On Large Genealogical Graph Layouts

Radek Mařík

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

*Abstract:* Classical ancestor trees, descendant trees, Hourglass charts, and their visual variants such as node-link diagrams or fan charts are suitable for assessment of people's relationships when one is focused on a particular person (the so-called main person) and his/her direct ancestors and descendants. Such tree-based representations miss a broader context of relationships and do not allow quick assessment of several interlinked families together. We propose utilization of directed acyclic graph visualizations with constraints specified by 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 datasets containing 1600 people (a private family tree collection) and 3000 people (an Egyptology database of officials from 4th, 5th, and 6th dynasty).

## 1 Introduction

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 [1, 2, 3]. 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 relationship exists [4, 3]. 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 [5] belong to a set of traditional tools implemented by a majority of freeware, shareware, or commercial tools, for example Gramps [6] or MyHeritage [7]. 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 the so-called main or center person). The main person is 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 graphical representation results in a triangular shape. Such a classical node-link tree representation wastes about one half of the drawing area. There are other more space-efficient representations such as fan charts or H-charts [8, 9, 10, 11]. 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 [6] 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.

However, the situation with family members 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 with edge crossing and preferences on node clustering [12, 13, 14]. The genealogical tools often do not provide such specialized visualizations. At present it is possible to use methods dedicated to a general graph layout. Hierarchical layouts are suitable for genealogical directed graphs, for example, implemented and provided by tools such as `dot.exe` (DOT) in Graphviz package [15] or yEd [16]. 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. Based on our own experience and observations made during our cooperation with Egyptologists, the researchers prefer grouping based on families.

Fig 1 depicts Nyankhkhnum's and Khnumhotep's family reconstructed from the database of the Egyptian officials [17]. In this case, the layout was produced using the yEd tool. Although it is possible to improve such a layout manually, one cannot waste time redoing the layout for all database families whenever the database is updated.

It is possible to group children or their parents (but not both). Unfortunately, directed hierarchical drawing methods such as the very good one implemented as `dot.exe` [18] results in layouts with mixed generations

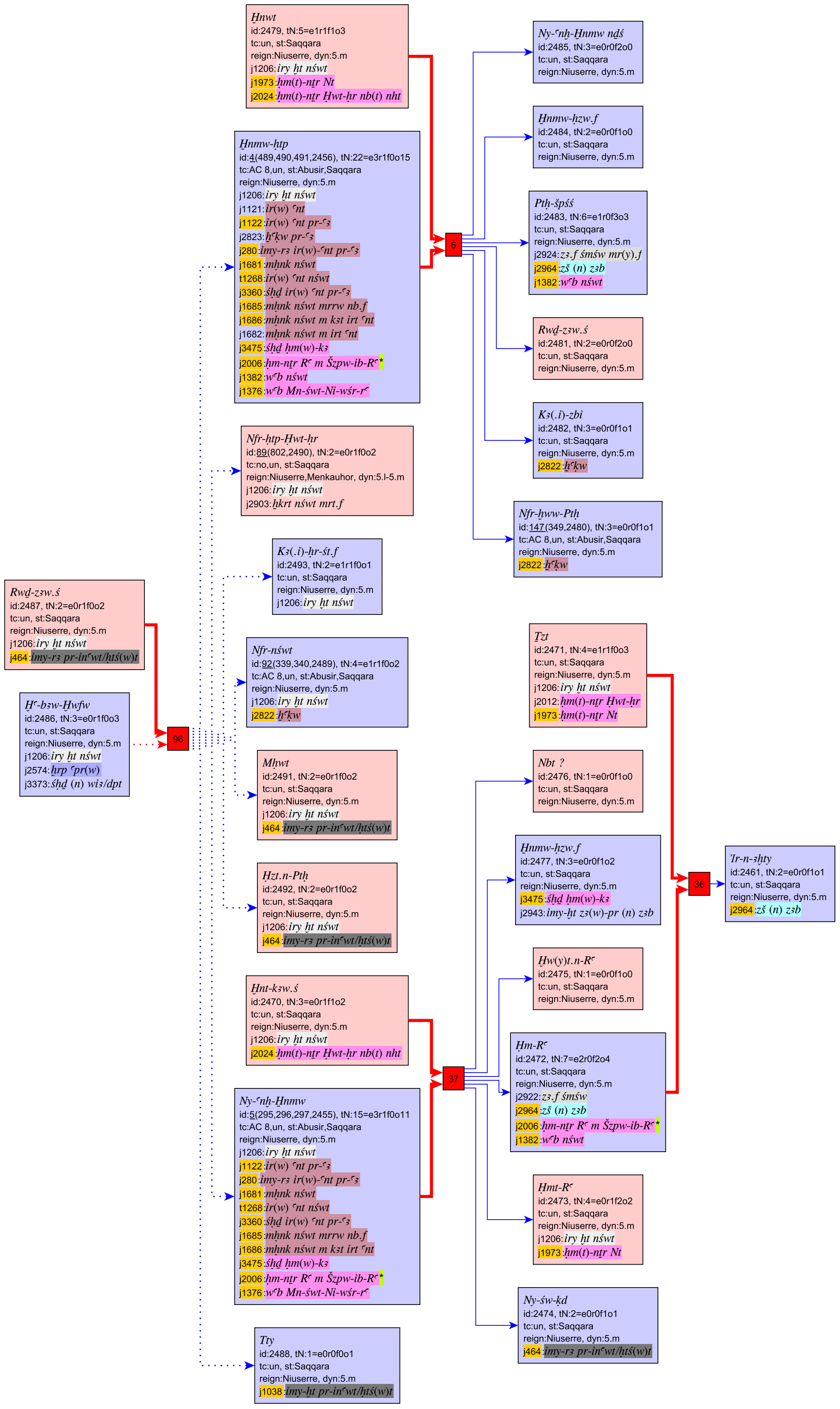

Figure 1: A family tree component presented using a tree layout which is illustrative of Nyankhkhnum's and Khnumhotep's family. The people rectangles contain additional information such as their titles.

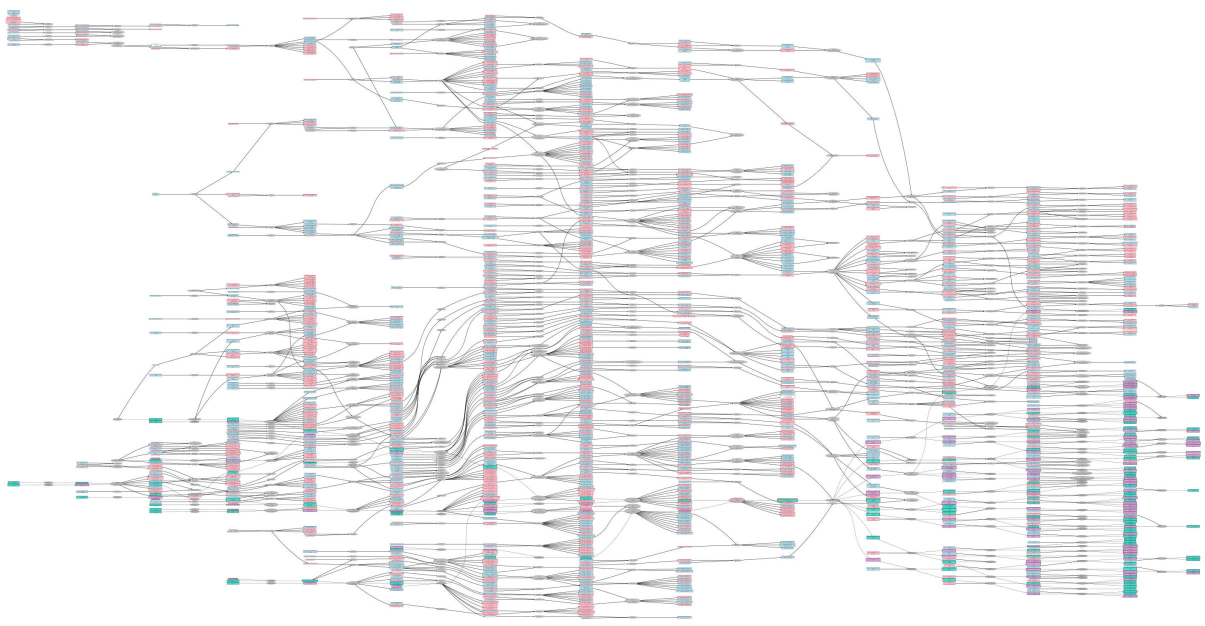

Figure 2: A sample private family tree consisting of 1671 people as rendered using the DOT tool without any further constraints. Colored rectangles represent people (reddish women, blueish men). Ovals capture their marriages. Although the visualization seems to be correct, there are many cases when people are moved into different generation layers and many children from different families are mixed. The quality of the picture is decreased to keep family privacy.

and groups mixing several families. Such layouts are difficult to read and comprehend. We are not aware of any method that would enable the definition and use of the necessary constraints. In this paper we focus on several principles that allow the determination of such constraints and how such constraints can be managed. 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, we focus on two most critical aspects discussed in [13] and dealing particularly with the first two steps of the approach proposed in [18]: 1/ determination of generations (layers, node ranks), and 2/ enforcing family grouping based on propagation of children and marriage orders through generations. We propose several approaches for handling such aspects and we provide efficient algorithmic solutions for them. Of course, one can consider other aspects as well. In this paper we focused only on these two.

The rest of the paper is organized in the following way. In the next section we provide an algorithm that allows setting ranks of nodes for an acyclic graph representing a traditional representation of family tree using marriage nodes. In the next section we design a method that allows propagation of children and parent ordering across generations (ranks). Finally, we discuss some results achieved if the constraints are mapped into DOT language and tested on datasets with thousands of nodes.

## 2 Ranking of Genealogical Graph Nodes

Even a DOT graph specification does not contain any constraints on node layers. Its implementation ranks nodes as proposed by many authors [13, 18]. In many situations the result layout is produced as required, see Fig 2. Unfortunately, the general criterion used in the DOT implementation leads to node placement breaking generation layering as it is usual and expected in genealogical graphs, i.e. children of one family at the same level and similarly their parents. The DOT language enables the specification that a subset of nodes shares the same rank. The majority of algorithms computing ranks are derived from the topological order computation ($O(n)$ time complexity) [19] and select one of many possible solutions that satisfy layer intervals of node placements. In this section we present an algorithm, using which the ranks of nodes can be determined for any genealogical graph. 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. 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 the following algorithm we assume that the processed graph is directed and acyclic. 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. Let us use a convention that generation layers are identified by numbers $\lambda(v)$ and successors have higher levels. Each node is assigned an interval of generation levels at which the node can appear with regard to a base level. The following proposed 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.

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

Thus, the node(s) with the lowest level can be determined. A generation 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.

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

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 generation 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. Any layout with nodes placed in layers following, for example, increasing generation levels always has the same direction of all edges. The edge layout direction cannot be reverted ever as it might occur in methods based on a general optimization criterion such as the one used in the DOT.

The algorithm uses two DFS passes with linear complexity. Therefore, the time complexity is $O(N)$, where $N$ is the number of graph nodes. Two arrays are used for the maintenance of minimum and maximum levels for each node. A DFS pass requires an implicit or explicit stack. Different implementations of the stack, a graph representation, and the related DFS implementation can result in different space requirements ranging from the maximum depth of the acyclic graph (its diameter) to the number of all nodes. The length of the queue in our implementation is constrained by $O(d * b)$, where $d$ is the maximum depth of the graph and $b$ is the maximum branching factor. Both $d$ and $b$ parameters do not cross value 15 in the majority of cases (the maximum number of generations, the maximum number of children/partners). Thus, the space complexity is again in the range of $O(N)$.

## 3 Same Generation Nodes Ordering

Using the state of art of graph layout techniques such as those implemented in Graphviz [18] leads to results that are almost acceptable, however, with some drawbacks. Assuming that a genealogical graph is layered according to the generation levels determined by the algorithm proposed in the previous section, the main complaint stems from mixing of children/partners from different families. 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. Siblings from different families can be mixed.

In this section we support the approach when siblings of one family are 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 when it 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. 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. We would need other constraints to resolve them. In this paper we provide only a simple solution based on a random order of families. As these cases are not common, the resulting edge crossing is acceptable. A more sophisticated solution would create three sets of marriage nodes. The middle set, consisting of nodes representing marriages of children from both families and determining the order of families, in a way minimizes edge crossing. The other two side sets of marriages can follow the order of the two families and the order of their children. Nevertheless, the general situation with more than one marriage involving two and more families is rather complex and is considered beyond the scope of this paper. We denote the defined order of children and spouses as basic order subsequences.

The proposed solution is based on a propagation of basic order subsequences from lower levels of generations to higher ones, and similarly in the opposite direction. A linear graph composed of a disjoint sequences of nodes belonging to a given generation layer is maintained. That means, at a particular step of the algorithm, the set of nodes belonging to the processed generation layer is decomposed into a set of linear sequences. Each sequence determines an order of its nodes that is kept unchanged. In each propagation step, the nodes of a sequence in one generation layer are projected into their successor/predecessor nodes in the next/previous generation layer. The resulting sequence is fused from sequences already defined in the next/previous generation layer. In fact, any contradictory order requirements leading to loops must be dropped. We are aware that more sophisticated techniques of such requirements dropping can be implemented and can lead to better layouts. Nevertheless, our present basic solution uses a strategy adding additional order constraints in a step by step manner. If a requirement would create a loop, it is dropped.

As the genealogical graph is assumed to be acyclic and connected, the shortest trail linking any two nodes can be found. Any triple of nodes spouse-marriage-spouse or child-marriage-child defines the order of the two nodes. As any two nodes in a given generation layer can be ordered, a single sequence of totally ordered nodes in each single layer can be created. In other words, basic order subsequences fully specify a topological order of all nodes in the graph. Of course, different layouts can be achieved if we select a different dropping criterion of redundant order requirements.

Let us describe a propagation technique using just subsequence structures. Initially, the sequence of siblings based on their birthdates belonging to a family is computed for each family with children. Similarly, a sequence of marriage nodes is created for spouses with multiple marriages. Then the process iterates from lower to higher generations. In each iteration all edges of sequences from the lower generation are propagated to edges linking the related sequences in the higher generation.

It is obvious that the critical operation is the mapping from nodes to sequences and linking of sequences. There are several possible solutions. Firstly, a given generation layer of nodes can be represented as a directed graph. Whenever we need the first or the last node of a sequence to which a given belongs we can find it through a path against or along the direction of edges, respectively. As sequences get longer, the processing time of this operation grows exponentially. Secondly, it is possible to maintain a mapping from each node to its sequence first and last nodes. Initially, each node references itself as the first and the last node of a primitive sequence consisting of the node itself. Whenever two sequences are merged, all its node references of the first and the last nodes must be updated. At present, our implementation uses this approach. We do not perceive any performance issues if used on graphs with several thousands of nodes. Thirdly, as merging of sequences can be considered as a union of two sets, the very efficient union-find algorithm can be used. Furthermore, we would need to maintain a reference to the first and last nodes for each such union sequence representative node. We will describe this efficient method further in this section.

A special treatment must be paid to linking of sequences. It is very easy to create a loop, for example, if there are two families, one with two boys and one with two daughters, and they create two families when the older boy is married with the younger daughter and the younger boy is married with the older daughter. In such a case we have contradictory requirements for the order of marriage nodes of young couples. If all such order requirements are propagated, a loop in the order sequences is created. At present we propagate an order requirement only if it does not create a loop. Loops can be created over a merged sequence or over the input sequences. All possibilities must be checked and avoided.

An actual efficient implementation of the propagation method is not complicated, and it is rather simple using a union-find technique and a binary tree. A sequence of nodes is projected into the other sequence through order edges linking subsequent nodes in the source layer. The resulting destination sequence of nodes must be decomposed into subsequences already existing in the destination layer. We can employ a combination of two techniques, the union-find method with its fast searching for a subsequence (set) representing node and a binary tree structure that is able to represent a subsequence as the preorder of its leaves and to accomplish two subsequences merging by adding a new binary tree root referencing the tree roots of subsequences as its children ($O(1)$ time). In other words, the union-find structure maps the graph layer nodes into their current maximum subsequence tree roots ($O(\alpha|V|)$, where $\alpha$ is inverse Ackermann function [19]) and the selected binary tree roots are then merged. Thus, processing of any graph node can be performed in almost constant time. The algorithm must make three passes through all layers of the genealogical graph, i.e. each constraint must be propagated fully in both directions. Thus, the overall asymptotic amortized time complexity is $O(1+\varepsilon)$. It should be noted that subsequence merging using a binary tree does not suffer from possibilities of creating loops as each graph node is referenced just once and the binary tree node always represents a properly oriented subsequence.

## 4 Implementation, Experiments, and Discussion

We have not attempted to implement a completely new acyclic genealogical graph layout algorithm. We precom-

```
{ edge[style=invis]; node[style=invis]; "p0"->"p1"->"p2"->"p3";}
{ rank = "same"; "p0"; "I1436"; "I1221"; "I1140"; "I1073"; "I1141";}
{ rank = "same"; "p1"; "F0417"; "F0497"; "F0405"; "F0414";}
{ rank = "same"; "p2"; "I1185"; "I1417"; "I1224"; "I1236"; "I1152"; }
{ rank = "same"; "p3"; "F0477"; "F0415"; "F0413"; "F0475"; }
```

Figure 3: A snippet of a graph specification controlling node ranks.

pute the constraints on generation layers and node orders in each generation. These constraints can be mapped into a graph specification of some already implemented tools. In particular, the DOT specification implemented by Graphviz tools enables such extensions.

Constraints on generation layering can be mapped easily to rank directives of the DOT language. A special node is created for each generation layer. The several additional subgraphs are generated. The first subgraph determines the sequence of generation layers using special generation nodes. Both nodes and edges can be set with the attribute `style=invis` so that these nodes and edges are not shown in the generated drawing although they control the layout. Then, other unnamed subgraphs are generated for each generation layer with the attribute `rank=''same''` as a list of node identifiers belonging to that generation prepended with the node identifier of the given generation layer. A snippet of such an additional DOT specification is shown in Fig 3. The snippet also demonstrates how generations of people with node identifiers starting with “I” are interleaved with generations of marriages nodes starting with “F”.

We selected two datasets for an evaluation of the proposed constraints contribution. The first dataset consists of 1671 people of the author’s 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 the year 1647. The second dataset consists of 3057 people of the database created by Egyptologists [17]. The database covers high rank officials from the $4^{th}$, $5^{th}$, and $6^{th}$ 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 help greatly Egyptologists to assess quickly investigated social phenomena.

The graph of the entire private family database was depicted on Fig 2. The layout was generated by DOT tool when the graph specification contains only a description of nodes representing people and marriages and genealogical edges (links between partners and their marriages, links between marriages and children). Although the overall appearance of the graph seems to be correct, there are serious deficiencies. Some parts of generations were moved upwards or downwards. Thus, the generations are mixed. In many cases, members of different families are mixed or some children are placed with a different family even if it causes obviously more edge crossing or longer edges.

When the constraints on generation layers are specified, the DOT tool might create a layout holding the rank specifications as depicted on Fig 4. One can spot immediately families as ovals followed by several rectangles. Not only family members are close to each other, but also their partial family trees are close, too. Unfortunately, one can also identify heavy crossing among spouses from several families with more children on the right side of the graph. There is always a marriage couple linking two large families together. An appropriate constraint avoiding such phenomena was proposed earlier in this paper. However, it must be properly combined with the computation of the constraints for children order and marriage order. Also, the solution must deal with a set of families that might be linked in a pairwise manner. At present, we are experimenting with several techniques to propose their best combination.

Experiments with families of the Egyptian database did not exhibit any breaking of these specifications as the families are quite simple and not larger than 50 family members.

A layout generated using the constraints proposed in this paper only without any further influence of the DOT tool is shown in Fig 5. The ranks of nodes were placed uniformly in a horizontal direction while their nodes were placed uniformly in a vertical direction. The nodes were linked with straight-lined edges. The layout is created very quickly (below 0.5 second with a Python script on DELL XPS 13 using an Intel i7 2GHz processor.

Nevertheless, there are also other issues connected with the DOT tool. The DOT tool takes the proposed order of nodes only as initial advice that does not need to be followed. Thus, if the DOT implemented criterion produces stronger values, it can break the specified node order and the layout can be again very confusing. For example, several ranks might be merged to save space if a generation layer is sparse. One might link children of a family with directive `subgraph`, but there is no specification on how ranking and subgraph specifications are combined and how they worked together. We performed a number of experiments with such more complex combinations with rather unpleasant results. We are not aware of any other tool that would allow a specification of a graph where one part controls the layout and the other part is presented, i.e. only coordinate positions of nodes are computed and edges are routed.

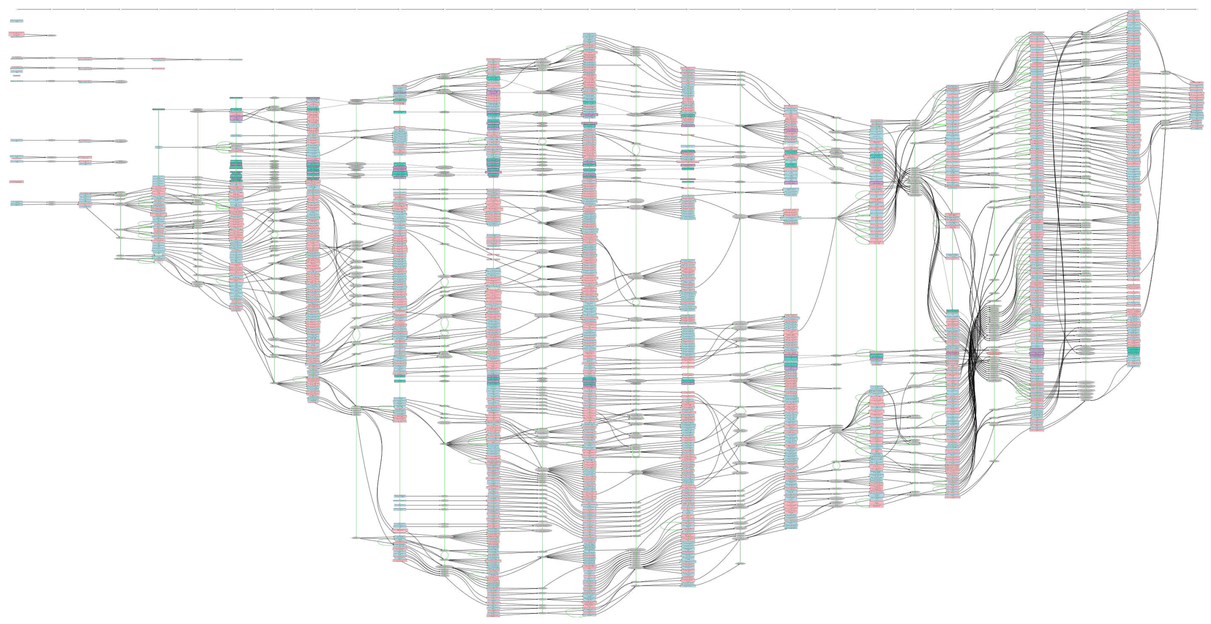

Figure 4: A visualization of the sample private family tree consisting of 1671 people if it is rendered using the DOT tool with the constraints on node ranks and their order within their ranks. Green edges control the layout. The top sequence of nodes defines ranking/generations. The quality of the picture is decreased to keep family privacy.

## 5 Conclusion

In this work we proposed two simple constraints on node order with regard to their ranks and to their order in ranks. The constraints produce graph layouts that are more acceptable for the user if they deal with large family trees combining several trees into a single acyclic graph. 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 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 proposed constraints do not cover properly a situation when more families with many children and a larger number of their mutual marriages are involved. Some hints on a better treatment were provided, but the search for their best combination is the current subject of our research. The proposed approach performs well if genealogical data resembles a composition of structures similar to trees with occasional crossovers of large families with many children.

## Acknowledgement

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

## References

[1] W. T. Tutte, "Convex representations of graphs," *Proceedings of the London Mathematical Society, Third Series*, no. 10, pp. 304–320, 1960.

[2] ——, "How to draw a graph," *Proceedings of the London Mathematical Society, Third Series*, no. 13, pp. 743–768, 1960.

[3] H. Gibson, J. Faith, and P. Vickers, "A survey of two-dimensional graph layout techniques for information visualisation," *Information Visualization*, vol. 12, no. 3-4, pp. 324–357, 2013. [Online]. Available: http://ivi.sagepub.com/content/12/3-4/324.abstract

[4] C. McGrath, J. Blythe, and D. Krackhardt, "Seeing groups in graph layouts," *Connections*, vol. 19, no. 2, pp. 22–29, 1996.

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

[6] (2016) Gramps. genealogical research software. https://gramps-project.org/. Accessed: 5.6.2016.

[7] (2016) Myheritage. https://www.myheritage.cz. Accessed: 5.6.2016.

[8] C. Tuttle, L. G. Nonato, and C. Silva, "Pedvis: A structured, space-efficient technique for pedigree visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 16, no. 6, pp. 1063–1072, Nov 2010.

[9] V. Yoghourdjian, T. Dwyer, G. Gange, S. Kieffer, K. Klein, and K. Marriott, "High-quality ultra-compact grid layout of grouped networks," *IEEE Transactions on Visualization*

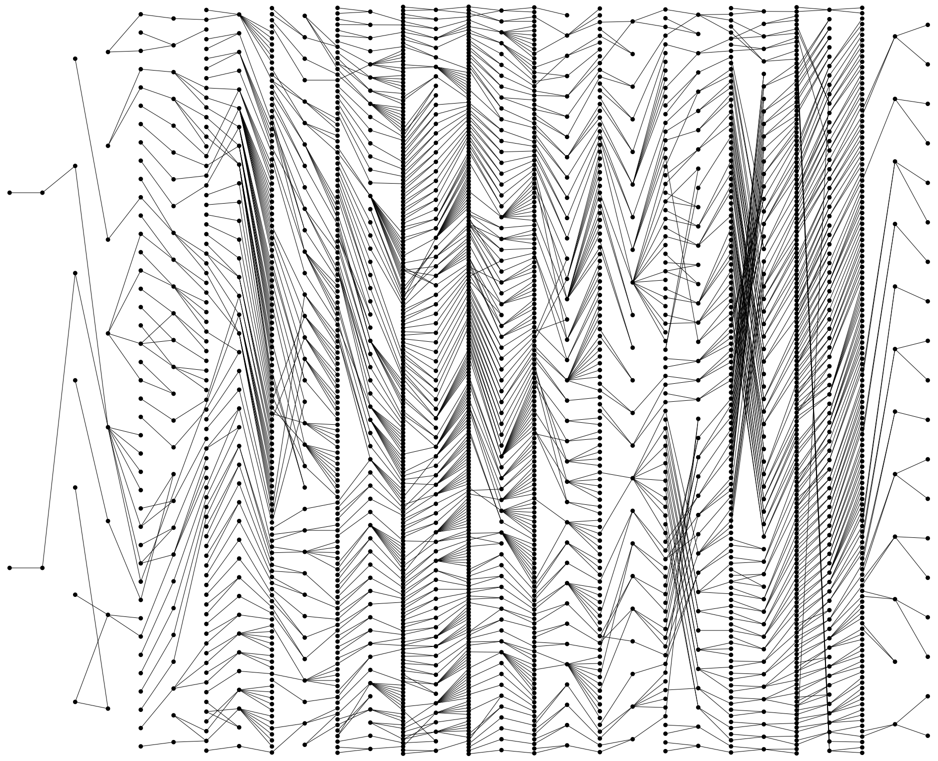

Figure 5: A visualization of the sample private family tree created from the rank and node order constraints proposed in this contribution only. An ideal layout would result in edges creating "waves" only. The heavy edge crossing on the right side is caused by the too simple local dropping of contradictory order requirements.

*and Computer Graphics*, vol. 22, no. 1, pp. 339–348, Jan 2016.

[10] R. Ball and D. Cook, "A family-centric genealogy visualization paradigm," in *14th Annual Family History Technology Workshop*, Provo, Utah, 2014.

[11] S. Kieffer, T. Dwyer, K. Marriott, and M. Wybrow, "Hola: Human-like orthogonal network layout," *IEEE Transactions on Visualization and Computer Graphics*, vol. 22, no. 1, pp. 349–358, Jan 2016.

[12] J. N. Warfield, "Crossing theory and hierarchy mapping," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 7, no. 7, pp. 505–523, July 1977.

[13] K. Sugiyama, S. Tagawa, and M. Toda, "Methods for visual understanding of hierarchical system structures," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 11, no. 2, pp. 109–125, Feb 1981.

[14] K. Sugiyama and K. Misue, "Visualization of structural information: automatic drawing of compound digraphs," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 21, no. 4, pp. 876–892, Jul 1991.

[15] (2016) Graphviz - graph visualization software. www.graphviz.org. Accessed: 5.6.2016.

[16] (2016) yed graph editor. http://www.yworks.com/products/yed. yWorks. Accessed: 5.6.2016.

[17] V. Dulíková, "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. dissertation, Charles University in Prague, Faculty of Arts, Czech Institute of Egyptology, 2016.

[18] E. R. Gansner, E. Koutsofios, S. C. North, and K. phong Vo, "A technique for drawing directed graphs," *IEEE Transactions nn Software Engineering*, vol. 19, no. 3, pp. 214–230, 1993.

[19] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Third Edition*, 3rd ed. The MIT Press, 2009.