Journal of Genetic Genealogy

Journal: www.jogg.info
Originally Published: Volume 10, Number 1 (Fall 2022)
Reference Number: 101.003

GRAPHS FOR GENEALOGISTS

Author(s): David A Stumpf, MD, PhD

Journal of Genetic Genealogy

GRAPHS FOR GENEALOGISTS

By David A Stumpf, MD, PhD

Abstract

Graphs for Genealogists (GFG) is an open-source software package with an application front-end, a Neo4j database and a plugin designed and optimized for genealogy data management and analytics. It loads family tree data in GEDCOM format, a set of consumer DNA test results, and genealogist curated files providing links between graphs. The primary purpose of the analytics is to discover new insights and provide actionable recommendations for further genealogy research. GFG traversals collect concatenated strings to create Ahnentafel numbers and enable filtering on X-linked inheritance and other patterns. Traversals from the family tree through DNA matches to chromosome segment data discover triangulation groups and monophyletic segments aligned with specific family tree branches. Graph algorithms from the Neo4j Graph Data Science plugin discover communities (clusters) aligned with family tree branches. Hierarchical trees include patrilineal and matrilineal trees, DNA haplotrees, ORDPATH enhanced renderings, and hybrids linking these together. Chromosome painting and 3D renderings help users interpret the results. Recommendations include manageable sets of persons from a pool of over 250,000 DNA matches. There are many opportunities for further development of graph analytics including a paradigm shift to using stable elements aligned with a specific family tree branch.

Background

Genealogists use graphs to chart connections of their relatives and the facts related to them. There are many excellent and well documented tools available. Most are based on relational algebra or matrix mathematics. A few utilize graph analytic and visualization methods with data stored in structured files (e.g., csv, json, etc.) assembled by the user or other tools. The flexibility and scalability of these methods is constrained by the computational challenges, exemplified by the large sparse matrices in clustering tools.

Graph theory evolved from the mathematical contributions of Leonhard Euler in the 18th century. Modern connected systems, most notably the internet, stimulated the dramatic proliferation of graph data management systems and methods. “Thinking is graphs” is a theme in this paper suggesting the need for a new world view that is intuitive but not yet prevalent in genealogy. Graph thinking is transformative, encouraging the emergence of novel insights and methods. This article seeks to facilitate that transformation for genetic genealogists.

Neo4j is an industry leader in native graph databases[1]. Neo4j developed and uses the Cypher query language[2] which is more intuitive and easier to use but based on the underlying Graph Query Language (GQL) standard[3]. Neo4j and Cypher are written in Java®. Several plugin packages are used to extend its capabilities. These include plugins available from Neo4j: Awesome Procedures for Cypher (APOC)[4] and Graph Data Science (GDS)[5]. Data types include familiar types (string, integer, date, etc.) and graph specific types such as nodes, relationships, and paths. Nodes and relationships may have properties which can be indexed. A query usually searches, using an

index, to find starting node(s) and then traverses the graph along relationships specified in the cypher query. This is much more computationally efficient that SQL queries and their joins.

Genealogists use many types of graphs. Linking different graphs is accomplished by adding relationships from a user curated file that links a node in one graph to a specific node in the other graph. For example, a person node from a GEDCOM file can be linked to a DNA kit and it, in turn, to a set of shared matches or DNA segments. Once these enhancements are in place, queries can rapidly traverse across several graphs. This strategy also creates flexibility because adding a new graph is easily done. The find and traverse query strategy is not only efficient[6, p. 222], but scalable[7]. A notable distinguishing feature of graph databases is the very minimal deterioration in performance as more data is added[6, pp. 11–13].

Methods

A Neo4j® v 4.x Enterprise database[8], installed on a local computer, is used in these studies. A new "Graphs for Genealogists" (GFG) plugin was created to further extend Neo4j capabilities and facilitate genealogy data management and analytics. The GFG PlugIn (GFG-PI) is an open-source set of user defined functions (UDF) coded in java v 11 to seamlessly interface with the Neo4j API-driver (Appendix 1). A UDF is called, sometimes with user entered parameters, from the Neo4j browser or an optional GFG software package (GFGS)[9] developed to facilitate the transition of genealogists from current to graph methods. The software prompts, instructions and this article guide the user through the steps of data preparation, loading the data and enhancing it. The GFGS uses two files to manage project data (Appendix 2); these must be in a specific Windows directory: c://Genealogy/Neo4j/. The most recent version of the GFG-PI java jar file is downloaded from GitHub[10]. Implementation requires minor additions to the Neo4j configuration file[11]. The GFG-PI was developed in Maven.[12] Developers can find the *.pom[13] and java class files[14] at GitHub. Once installed the functions are listed in the Neo4j browser with this cypher command which uses the "gen" prefix defining the GFG-PI:[i]

```
//genealogy user defined functions
Show Functions yield name, signature, description, returnDescription
where name STARTS WITH 'gen'
return name, signature, description, returnDescription
```

The GFGS assists users with prompts and instructions in the next several implementation steps. The user creates a project and enters or selects the parameters required for navigating to their files and authenticating access to the Neo4j local server. The Load Data menu has submenu items that are used in sequence. The first loads the GEDCOM and Family Tree DNA (FTDNA) data. Both must be loaded to continue. The second enhances the database and requires entry of the common ancestor (see below). The third loads the FTDNA mt- and Y-haplotree reference data.

The GFGS also facilitates the initial reporting. The Reports menu has submenus. The reports are Excel files with the report output and, at the bottom of the worksheets, the query(ies) statements, database used and other

[i] Cypher commands will use this distinctive formatting. Comment lines start with //.

explanations. The user can copy the query statement to the Neo4j browser and run it there, adjusting the parameters to explore the effects. For example, some queries use the range of centimorgans to filter the results. If the research concerns distance relatives the centimorgan values used would be less.

Individual GFG-PI functions are called from the Neo4j browser or GFGS. The UDF available for preparing this article are in Appendix 3. If run in GFGS, the user is prompted for parameters, if required by the UDF. In the Neo4j browser the command format begins with "return" and may include parameters, such as the three in this command for finding all matches mapping to chromosome 15 between positions 15,000,000 and 100,000,000[ii]:

gen.dna.matches_at_chr_region('15',15000000,100000000)

Neo4j schema requirements are specific for each project. There are several examples of schema for genealogy data but they are not optimized for genetic genealogy analytics[15],[16],[17]. Optimizing the schema was a major focus of the current research. The GFG-PI has functions to automate the extraction, transformation and loading (ETL) of files. The GEDCOM import creates Person, Union and Place nodes and relationships between them. The import of FTDNA csv files produces Kit, DNA_Match and Segment nodes and relationships between them. The GEDCOM, Kit and DNA_Match nodes are then connected using a curation file (Appendix 4) which is required to have the exact spelling of FTDNA match names and, when known, the GEDCOM record number and the FTDNA kit number. New matches of interest may not yet be in the family tree and therefore lack a GEDCOM record number. Kit numbers are not available for many matches. The ETL creates unique, single nodes for each Person, Union, Kit, DNA_Match, Segment and triangulation group (tg) nodes. Granular kit specific details are pushed to the match_segment and match_tg relationships, including properties for the source kit (propositus) and the match reported in the FTDNA chromosome browser csv file. Specifically, the properties are p for propositus and m for the match. Each row in the FTDNA chromosome browser results in a match_segment relationship. The ETL uses the curated triangulation group file to create the tg nodes and tg_seg, match_tg and person_tg relationships linking tg to Segment, DNA_Match, and Person nodes respectively (Appendix 5). The tg_seg relationship is set first and then used to add match_tg relationships when the DNA_Match node maps to a segment within the triangulation group. The match_tg relationships have granular property data similar to the match_segment relationship (see Appendix 6).

One-time enhancements to the graph, using existing data, reduce the length of traversal paths creating query efficiencies. For example, without an enhancement, linking a family tree to DNA segment data requires traversing the path Person→Kit→DNA_Match→Segment. Adding the Person record number to the Kit and DNA_Match nodes enables queries starting downstream from the unenhanced query. When the curation file permits, the GEDCOM record number is added as a property to the Kit and DNA_Match nodes.

Neo4j, beginning in version 4, supports indexes based on relationship properties. We enable filters by adding the GEDCOM record number to the match_segment relationships. This may include the record number for the propositus (p_rn) and the match (m_rn). A full text index is used for ancestor surname lists provide by some DNA matches[18].

[ii] User defined function call use distinctive formatting.

To support efficient queries focused on a specific ancestral line, the user chosen ancestor GEDCOM record number is added as a property to the Person and Kit nodes (ancestor_rn) and the match_segment relationship (p_anc_rn and m_anc_rn). Adding, deleting, or changing these properties is easy in comparison to longer traversals without them. Segment nodes of descendants of the ancestor has a property anc_desc, enabling queries to quickly find them.

Genealogical relationships are computed for two individuals using the GEDCOM record numbers and the results are added as properties to the match_by_segment and shared_match relationships, specifically rel (1C, H2C, etc), cor (coefficient of relationship), pair_mrca (common ancestor of the match pair), and gen_dist (genetic distance). The UDF **gen.rel.add_relationship_property** traverses the graph from the two individual's Person nodes to their common ancestor(s) Person node and uses the number of common ancestors and the path lengths from each Person node to look up the relationship in a set of reference fam_rel nodes using an index string concatenating two path lengths and the number of common ancestors. For instance, 3:4:1 is the H2C1R and 3:4:2 is the 2C1R. The UDF **gen.rel.relationship_from_RNs** returns more than one relationship if there are more than two common ancestor paths. For example, the author's great-great-grandfather married two sisters, producing ¾ siblings. A distant "cousin" is both H4C (5:5:1) and 5C (6:6:2); the former sharing a single great-great-grandfather and a prior generation great-great-great-grandparent couple. The fam_rel node set also contains, when available, centimorgan data from the Shared Centimorgan Project[19].

Shared matches are explicit in the Family Finder csv file from FTDNA and captured in the shared_match relationship. The match_by_segment relationship between two matches is created by aggregating the match_segment property data. This produces the match_by_segment properties described in Appendix 6. The x_gen_dist property is added to the match_by_segment relationship using the UDF **gen.dna.x_chr_min_genetic_distance**. FTDNA reports X-matches without any knowledge of the family tree. The x-inheritance genetic distance (x_gen_dist) property is 0 (zero) if there is no genealogical X-chromosome path between the matches; this property enables queries limited to relevant X-matches.

The UDF **gen.tgs.setup_tg_environment** is used to associate data to a user-designated ancestor of interest, such as a remote common ancestor of the project participants. It can be re-run to reset the ancestor association to a different ancestor. The ancestor_rn is added as a property to the Person, DNA_Match and Kit and subsequently used in associating segments with the ancestor. When re-run, the prior ancestor_rn properties are removed and replaced by the new ancestor_rn. The segment ancestor association is accomplished by 1) adding the ancestor_rn property to the Segments nodes, 2) a p_anc_rn and/or m_anc_rn property to the match_segment relationships, 3) by creating sorted and ancestor linked segments, linking them with a new relationship (seg_seq) and 4) adding an anc_desc property to these Segment nodes. The enhanced properties enable queries limited to ancestor associated elements.

The Segments linked to the descendants of a common ancestor are identified by the UDF *gen.dna.matches_by_segments_anc_desc*. The UDF adds the property anc_desc to the identified Segment nodes and match_segment relationship. The UDF ***gen.tgs.setup_tg_environment*** calls upon another UDF, ***gen.tgs.create_seg_seq_edges***, to create the seg_seq relationships. The Segment nodes from a descendent kit

and within a triangulation group are sorted by their chr and strt_pos properties and then linking by a seg-seq relationship with a tgid property, associating it with the triangulation group.

Data for the ETL includes a GEDCOM, several sets of Family Tree DNA (FTDNA) csv files, a curated triangulation group file and a curated file linking the GEDCOM and DNA results. The GFGS requires a specific directory structure for these files [20]. GFG-PI loads data using two methods. The LOAD CSV method[21] is efficient for small csv files and supports batch processing. For larger or slower loading datasets, the APOC LOAD CSV[22] function enables batch processing and recovery from server interrupts or locks. During the ETL the database should not be otherwise used to avoid locks which block the ETL processes. Neo4j cypher queries are most efficient when repurposing data already loaded. For example, after loading the Person and Union nodes you can use a cypher query to create the family relationships (father, mother, spouse) between two Person nodes[iii] and the child relationship between Person and Union nodes.

Indices are created to enable quick searches for start nodes where traversals begin. Indices are on single properties or compounded using multiple properties (Appendix 7). A full text index of ancestor lists enable Lucene searches[18].

GFG-PI includes separate ETL functions for downloading the FTDNA mt- and Y-haplotrees [23] and the HapMap[1] and loading them into Neo4j. The HapMap is loaded to a separate database named hapmap. It is used to compute the centimorgan in a chromosome region that is newly produced by analytics. During the ETL for a project, a set of chr_cm nodes is created with a node for each chromosome and its minimum start and maximum end position for the segments in the project. The HapMap is then used to compute the total chromosome centimorgan included in the project, adding the cm property with this value. The UDF *gen.dna.chr_portion_of_segment* is then able to compute the portion of the chromosome subsumed by a match_segment relationship. This is used to filter out segments larger that a specified portion (generally 0.5) which may otherwise confound chromosome analytics and painting visualizations. While the coefficient of relationship or cm can be used for this filtering, the former is not available for persons not in the GEDCOM and the portion of cm on a chromosome varies with the size of the chromosome.

The Neo4j browser display formatting is controlled by the grass file, analogous to a css file in HTML5. Grass files help standardize displays between users and expose properties appropriate for a specific visualization (Appendix 8). The **:style** command lists the current grass file content. External grass files are dragged and dropped in the Neo4j browser window, changing the visualization. Grass files enable displays of multiple properties, and custom colors and sizes for nodes and edges that are appropriate for the specific analysis.

The GFG-PI uses Ahnentafel numbers for ascending family trees. A base-2 Ahnentafel is created by assigning 1 to the propositus and then concatenating a 0 for fathers and 1 for mothers during tree traversals. The concatenated bitstring is used directly or converted to base-10, in either case enabling sorting in pedigree order. Descending family trees are more complex because there are often multiple children from a union. Publications typically use the NGS Quarterly System or Register system[25], but neither is optimal for computer sorting. The

[iii] match (p:Person) with p match (u:Union) where u.uid=p.uid with p,u match (f:Person) where f.RN=u.U1 with p,f,u merge (p)-[r:father]->(f)

GFG-PI uses ORDPATH to sort family lines in hierarchical order[26]. Creating the position within the generation is a UDF with two parameters, a cypher query and an integer for the position of the generation field in the query result. This is a generic solution which allows any cypher query to add the generational position.

Visualizations are important in graph analytics. The Neo4j browser renders graph images when queries returns only nodes and relationships. But the Neo4j browser rendering may require manual processing to create a useful image. Once finalized, a SVG file can be exported. Neo4j Bloom can be used to visualize and explore large graphs. Other tools are required to automate visualizations optimally formatted. Many segment reports produce files ready for bulk import into DNA Painter[27]. Some queries are included in reports which run in GraphXR[28] to produce 3-D visualizations. Neo4j APOC can export data formatted for Gephi[29]. Improved visualizations are a priority for future development of GFG.

Results

Data load times vary depending on the size of the GEDCOM and FTDNA files data files (which are differ between projects), the computer CPU speed, and available RAM and disk space. On a Window 10 64-bit PC with Intel(R) Core(TM) i9-10900KF CPU @ 3.70GHz , 64gB of RAM and a 1 TB internal drive the load times vary from a few minutes for 15 FTDNA kits with few matches up to 20 min for a large Stinnett surname project with 66 kits and their many matches. The Neo4j database for the Stinnett project was 812 mb in size and contained 1,332,118 nodes, 2,612,397 relationships, and 24,658,889 properties (Supplement 1, Appendix 9). These statistics can be used to compute the database size[30], but for GFG it is underestimated[iv] because of the size of the content of some GFG properties, most notably the list of ancestors submitted by matches. GFG-PI does not currently support updates to the GEDCOM or FTDNA files on the Neo4j server. With these load times the best practice is to re-load the data if there are updates to the GEDCOM or DNA files.

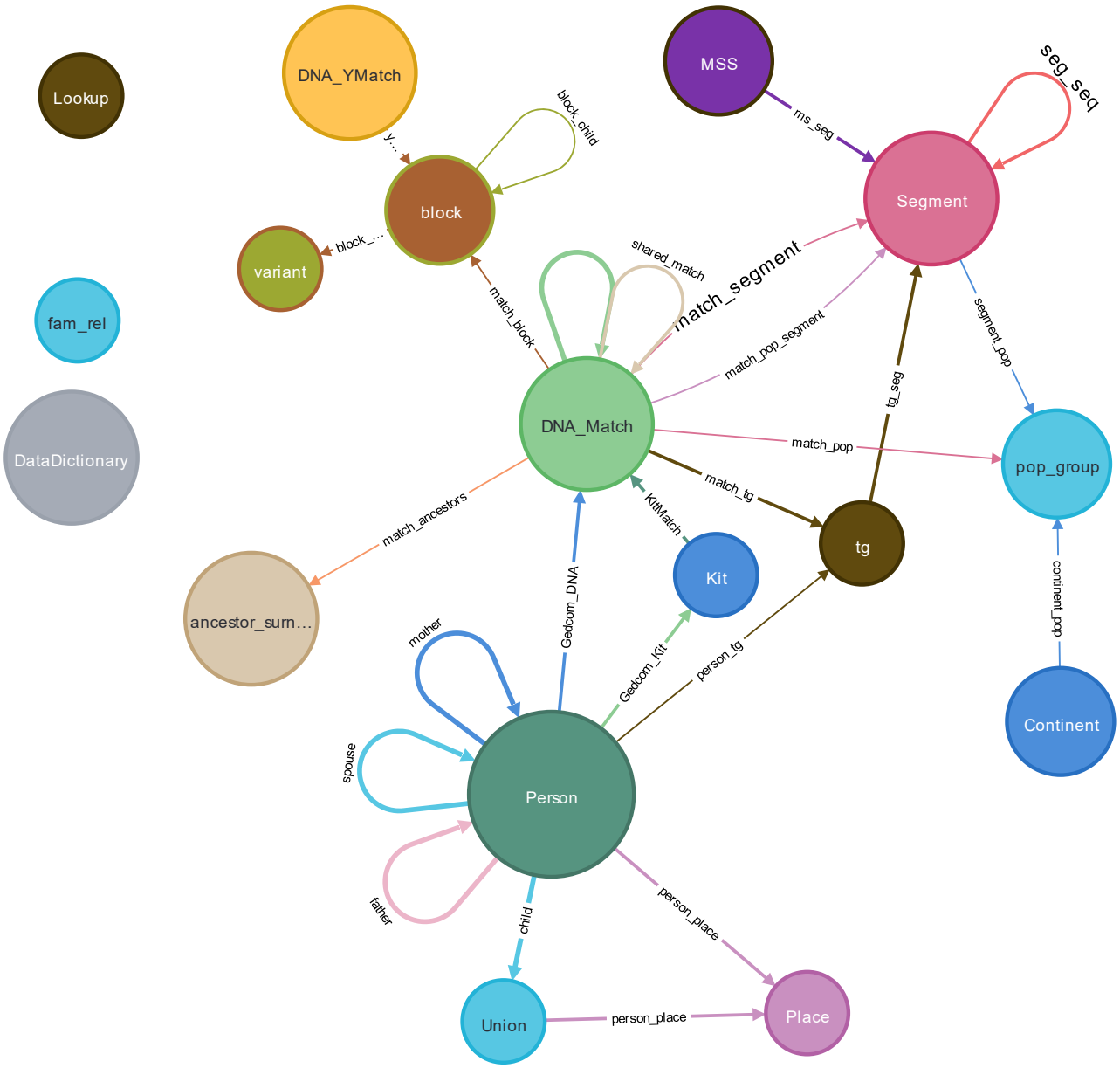
Supplement 1 shows the schema and data summary after all steps in the load sequence and enhancements. The data are from the Stinnett Surname Project at FTDNA. The 66 kits produced 247,437 DNA_Match and 356,721 Segment nodes. The latter two create 371,653 match_segment relationships. The other worksheets in Supplement 1 allow users to inspect the data and assess the quality of the ETL process.

The optimized Neo4j schema is illustrated in Figure 1. The data dictionary includes the element type (node or relationship), its name, properties, datatypes, whether it is required (e.g., always present) and a description explaining its use (Appendix 6). The schema indices are shown in Appendix 7.

A single GFG-UDF loads the GEDCOM, DNA and curated data: **gen.load.load_everything().** The user can observe the process by inspecting the Neo4j import directory to which the csv files are sequentially added and then loaded. The import directory tracking.txt file, best viewed in Notepad++[31], shows the progress with kit, kit name and, if available, the GEDCOM record number. This GFG-UDF sequentially calls upon other functions to load these data.

[iv] Estimate was 1gB for 3 clusters or ~333mB per cluster. GFG uses only one cluster.

Figure 1. The Neo4j Schema. The loops back to nodes form hierarchies such as family trees (father, mother) or haplotrees (block_child) The query creating this image is CALL db.schema.visualization()



The UDF **gen.workflow.initial_discovery()** runs a series of other UDFs to provide the user with a set of initial reports using default settings. An example "Know your data report" is Supplement 1. The in-common-with (icw) report has a row for every pair of DNA testers who are in the family tree and a list of all their icw matches and the shared centimorgans with each tester-match pair. The match_by_segment relationship links DNA_Match nodes using the raw segment data and recapitulates the shared match results reported by FTDNA. The cypher

query producing this report[v] illustrates a common graph strategy to identify shared entities: (node1)->[rel1]->(shared entity)<-[rel2]-(node2). Neo4j traverses from both node1 and node2 to the shared entity, in this case a shared match or segment. Enhancements to the graph always use filters for centimorgans ≥ 7 and snp counts ≥ 500 to minimize confounding by small segments[32].

The UDF for the cluster match report requires 3 parameters. For example, 3 as the minimum cluster size and 7 and 250 the centimorgan range. The query finds sets of testers whose number is 3 or more (a cluster) and then finds all other matches who match each and every member of the cluster[vi]. The query itself does not verify that all the newly identified matches match each other, but independent research does. Because the cluster members all share a designated ancestor, the odds are much greater that the identified matches are in the same family branch. Output files contain the UDF and all one-step UDF contain the cypher query. The user can re-run the UDF or cypher query with different parameters. Ranges with larger centimorgans will find more closely related matches. Ranges with small centimorgan ranges will identify more distant relatives or some matches that are identity by state or chance. The cluster match UDF also produces a field with a query which, when run, will produce a file suitable for uploading segment data to DNA Painter.

The Neo4j Graph Data Science PlugIn[5] is used in the GFG UDF **gen.algo.community_detection_icw**., accepting three parameters. The first parameter specifies the algorithm to run and the other two the centimorgan range. The UDF has four-steps. The first step creates a weighted virtual graph using the match_by_segment relationship and centimorgan ranges[vii] and the second step runs the specified algorithm: 1 = Louvain[viii], 2=

[v] match (m1:DNA_Match)-[r1:match_by_segment]->(icw:DNA_Match)<-[r2:match_by_segment]-(m2:DNA_Match) where m1.fullname<m2.fullname and m1<>icw and m2<>icw and r1.cm>=7 and r2.cm>=7 and m1.ancestor_rn is not null and m2.ancestor_rn is not null with m1,m2,case when icw.RN is null then icw.fullname else '*' + icw.fullname + ' [' + icw.RN + ']' end + ' {' + toInteger(r1.cm) + ', ' + toInteger(r2.cm) + '}' as fn with m1,m2,fn order by fn with m1,m2,collect(fn) as cicw with m1,m2,cicw where size(cicw)<=50 return m1.fullname + ' [' + m1.RN + ']' as match1,m2.fullname + ' [' + m2.RN + ']' as match2,size(cicw) as ct,cicw as in_common_with_matches

[vi] match (k:Kit) where k.ancestor_rn is not null with collect(k.RN) as krns MATCH (k1:Kit)-[r1:KitMatch]->(f:DNA_Match)<-[r2:KitMatch]-(k2:Kit) where k1.RN in krns and k2.RN in krns and 250>=r1.sharedCM>=7 and 250>=r2.sharedCM>=7 and k1<>k2 with f,apoc.coll.dropDuplicateNeighbors (apoc.coll.sort(collect(k1.fullname) + collect(k2.fullname))) as ck,apoc.coll.dropDuplicateNeighbors (apoc.coll.sort(collect(k1.RN) + collect(k2.RN))) as crn with f.fullname as fullname,size(crn) as ct,ck,crn with fullname,ct,ck,crn where ct>3 -1 with fullname,ct,ck,crn order by fullname with collect(fullname) as fn,ct,ck,crn with ct,ck,crn,fn with ct as Kit_ct,size(fn) as Match_ct,ck as Kits,crn as Kit_RNs,fn as Matches with Kit_ct,Kit_ct+Match_ct as Total,Kits,Kit_RNs,Matches where Total<50 with Kit_ct, Total,Kits,Kit_RNs,Matches return Kit_ct, Total,Kits,Kit_RNs,Matches order by Total desc

[vii] CALL gds.graph.create.cypher('icw','MATCH (m:DNA_Match) where m.RN is not null RETURN id(m) AS id', 'MATCH (m)-[r:match_by_segment]->(m2:DNA_Match) where 150>r.cm>25 RETURN id(m) AS source, id(m2) AS target, r.cm as weight',{readConcurrency: 4,validateRelationships:FALSE}) YIELD graphName AS graph, nodeQuery, nodeCount AS nodes, relationshipQuery, relationshipCount AS rels return nodes, rels

[viii] CALL gds.louvain.stream('icw', {relationshipWeightProperty: 'weight', includeIntermediateCommunities:true , tolerance:0.0000001,maxIterations:10,maxLevels:10 }) YIELD nodeId, communityId, intermediateCommunityIds with case when gds.util.asNode(nodeId).RN is not null then '*' + gds.util.asNode(nodeId).fullname else gds.util.asNode(nodeId).fullname end AS name, communityId as cid, intermediateCommunityIds as ici with cid,ici,name order by name with cid,ici,collect(name) as names

modularity optimization and 3=page propagation. The results are displayed in an Excel workbook (Table 1). The communities identified conform to branches in the family tree. There is a similar set of algorithms in **gen.algo.community_detection_shared_matches** in which the shared_match relationship is used. The third step extends the analysis using two other UDF and appends other information to the report. Using UDF **gen.rel.mrca_from_cypher_list** it finds the most distant common ancestor (MDCA), if there is one, for the set of DNA matches in the community. The UDF ***gen.mss.mss_data*** extracts the segments of the DNA matches in the community, the associated monophyletic segments (discussed below) and appends their counts, overlap counts and a list of the MSS segments in the community to the Excel file report. Finally, the fourth step creates a network visualization query suitable for GraphXR (Figure 2) and a separate csv file suitable for loading to DNA Painter, which visualizes the segments with different colors for each community. An additional query is produced for each specific community identified by the graph algorithm; it renders a GraphXR visualization of a DNA Circle (Figure 3) and its connection to segments (Figure 4). However, there is overlap between communities, as seen in Figure 2, and therefore the full topology of matches and segments in not seen in Figure 3 and Figure 4 This is particularly important because robust triangulation may depend on segments shared by two communities. To accommodate this conundrum, GFG uses a more complex query which re-samples community DNA_Match nodes for any other nodes (in a different community) which triangulate on the community's segments. This query also incorporates the *apoc.node.degree* function to limit the results to segments with at least three match_segment relationships (e.g., 3 DNA_Match node connections) and DNA_Match nodes with at least three match_by_segment relationships. While this this removes extraneous segments, it does not assure that the three segments all overlap.

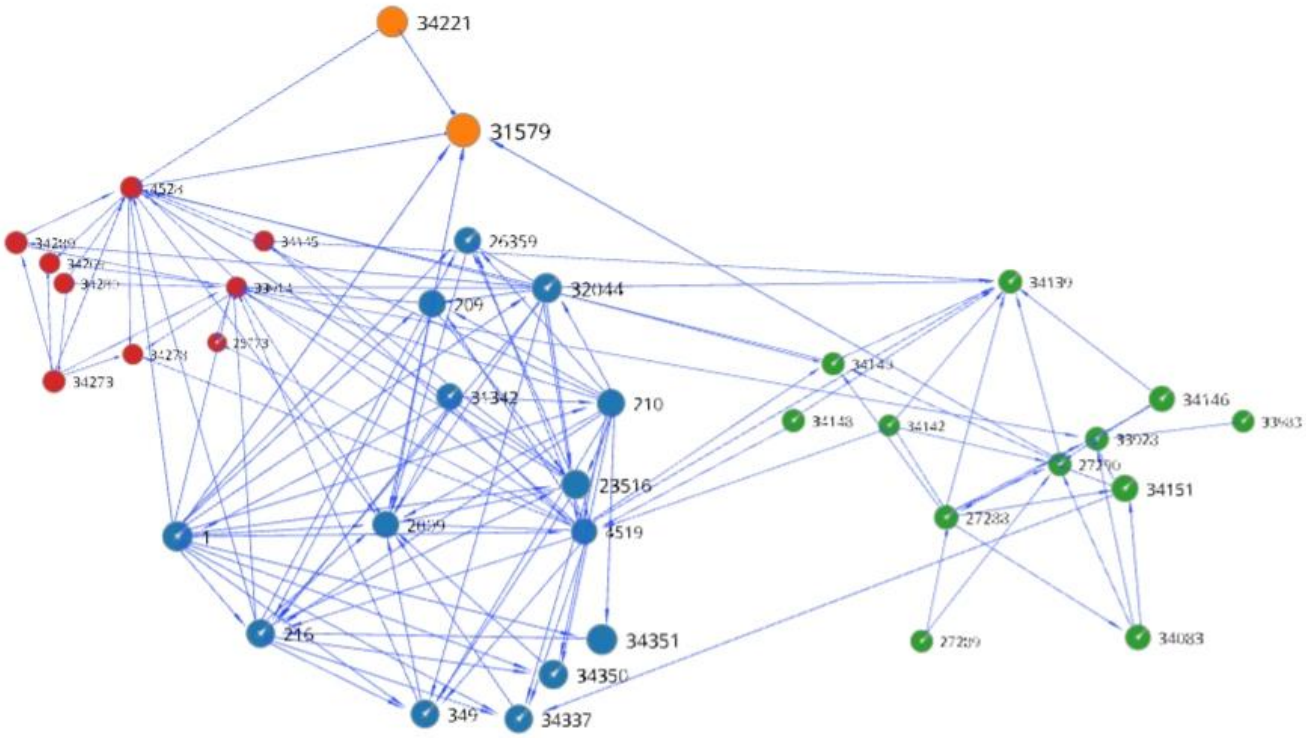
Using a wide range for centimorgans will create communities with a broader range of relatedness. This may then produce intermediary communities that can be used to create a dendrogram reflecting the family tree branches.

Table 1.. Teves project Louvain community detection algorithms identifies 6 communities. The community_id has no intrinsic meaning. The output is limited to DNA_Match nodes where the individual is also in the GEDCOM. Some of the individuals in the GEDCOM are not yet positioned in the family tree and are subjects of traditional genealogy research.

community	ct	matches
15	14	[1,216,349,2009,4528,23516,26359,31579,32044,33914,34221,34337,34350,34351]
5	7	[34083,34140,34145,34151,34273,34280,34289]
17	3	[209,210,4519]
18	3	[27288,33928,33983]
21	3	[27290,34139,34142]
8	2	[31342,34148]

with cid as community,ici,size(names) as ct, names as matches order by ct desc with community,ici,ct,matches where ct>2 return community,ici as intermediary_communities,ct, matches

Figure 2. Teves family Louvain communities are created algorithmically using the graph toplology and visualized in GraphXR. The communities (clusters) each have a distinct color.

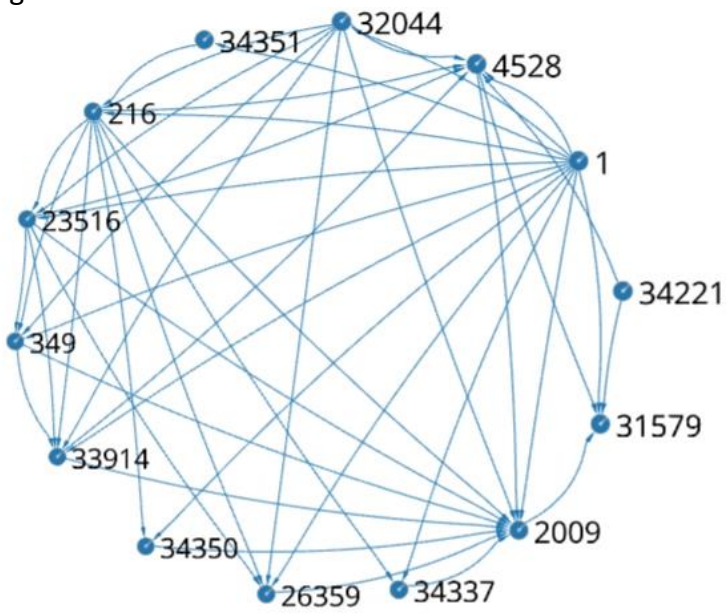


The double cousin report uses a cypher query[ix] to find family tree members who share 4 grandparents, aggregating all the double cousins by the grandparents. The query traverses to common ancestors two hops up the tree, aggregates the grandparents and then filters to select only those individuals sharing 4 grandparents. Then, it aggregates the sets of grandparents to get all their grandchildren who are double cousins. This query runs efficiently in Neo4j to discover, in the author's family tree database, 22 sets of four grandparents who collectively have 188 double cousins. Another query[x] returns all the combinations of pairs of double cousins; in the author's family this is 934 pairings.

[ix] Match (p:Person), (q:Person) match path=(p:Person)-[:father|mother*..2]->(CA)<-[:father|mother*..2]-(q:Person) with p, q,CA where p.fullname=replace(p.fullname,'MRCA','') and q.fullname=replace(q.fullname,'MRCA','') with p.fullname + ' [' + p.RN + '] (' + left(p.BD,4) + '-' + left(p.DD,4) + ')' as Name1, q.fullname + ' [' + q.RN + '] (' + left(q.BD,4) + '-' + left(q.DD,4) + ')' as Name2, CA.fullname + ' [' + CA.RN + '] (' + left(CA.BD,4) + '-' + left(CA.DD,4) + ')' as MRCAs with Name1,Name2,MRCAs order by MRCAs with Name1,Name2,collect(MRCAs) as mrcas, count(*) as MRCA_Ct where MRCA_Ct>3 with apoc.coll.dropDuplicateNeighbors(apoc.coll.sort(apoc.coll.flatten(collect(Name1) + collect(Name2)))) as double_cousins, mrcas return distinct mrcas as grandparents,size(double_cousins) as ct,double_cousins order by grandparents

[x] Match (p:Person), (q:Person)
match path=(p:Person)-[:father|mother*..2]->(CA)<-[:father|mother*..2]-(q:Person)
with p.RN as RN1,p.fullname as Name1,q.RN as RN2,q.fullname as Name2, count(*) as MRCA_Ct

Figure 3. GraphXR visualization of a Louvain community showing DNA_Match nodes arranged in a circle. The match_by_segment (ICW) relationships link many together. .



The project surname query finds all matches with the project surname specified by the name of the project in GFGS. The report names the matches and includes their matches by segment and the common ancestor of the matches who are in the family tree. The report includes previously identified triangulation groups if the segment falls within them. This report not only identifies new matches of interest but also may suggest new triangulation groups or updated ranges to add to the curated file.

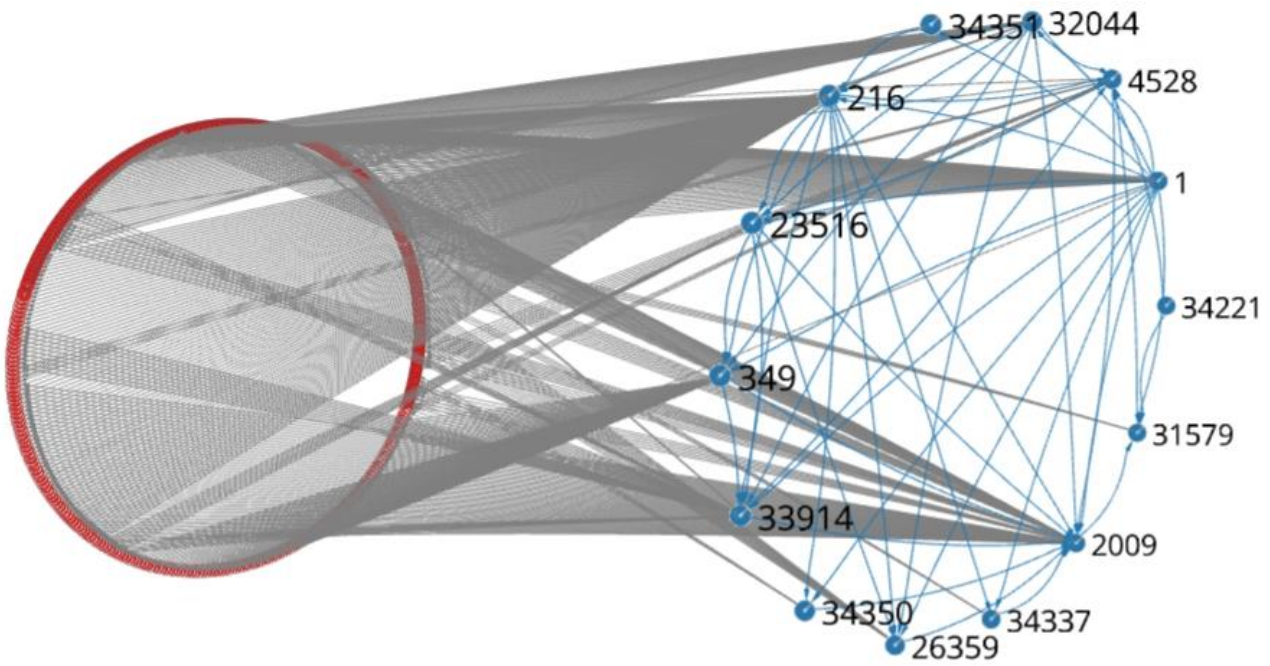
The ancestor descendant report produces an Excel workbook with three worksheets. The queries start at the ancestor specified as a UDF parameter. The first worksheet, analogous to ancestry.com ThruLines®[33], simply lists the ancestor, the descendant DNA tester, the number of generations to the ancestor and the path with names of ancestors at each generation[xi]. The second worksheet lists the segments with the descendants of the

where MRCA_Ct>3 and p.fullname=replace(p.fullname,'MRCA','') and q.fullname=replace(q.fullname,'MRCA','')
return distinct RN1,Name1,RN2,Name2,MRCA_Ct order by RN1

[xi] match (n:Person)-[z:Gedcom_DNA]->(m) with collect(m.RN) + collect(n.RN) as DM optional match path=(p:Person{RN:33454})<-[:father|mother*0..15]-(q:Person) where q.RN in DM with p,path,collect(last(nodes(path))) as cEnds optional match (q:Person)-[r:Gedcom_DNA]->(s:Person) where q in cEnds with r,p,[m in cEnds|m.fullname + gen.neo4jlib.RNwithBrackets(m.RN)] as E,[n in nodes(path)|n.fullname + gen.neo4jlib.RNwithBrackets(n.RN) + ' (' + left(n.BD,4) + '-' + left(n.DD,4) + ') '] as N return distinct p.fullname + gen.neo4jlib.RNwithBrackets(p.RN) as MRCA, E as Descendant_DNA_Tester,size(N) as generations, N as Path_to_Descendant_Tester

ancestor and matches at the segments. The final worksheet lists the known triangulation groups, the ancestor descendants who have DNA test results and a list of their most recent common ancestors[xii].

Figure 4. GraphXR visualization of a Louvain community DNA circle with its match_by_segment relationships (blue) connected by match_segment relations hips (gray) to associated segments (red).



ORDPATH is a concatenated bitstring produced from "Dewey order strings" by the UDF **gen.graph.get_ordpath**. The Dewey order string is a list of GEDCOM record numbers collected during the tree traversal (Table 2). ORDPATH was developed using base-2 at Microsoft and a hexadecimal version is the hierarchyId datatype in SQL Server and is also used by GFG-PI. ORDPATH is useful in sorting patrilineal, matrilineal and descendancy trees, such as an X-chromosome descendancy tree in which the generation count determined the indentation (Supplement 2). Renderings of mt- and Y-haplotrees also use this strategy and a Dewey string from a sequential list of the haplotree branch mutations. Another UDF, ***gen.graph.add_descenant_order***, processes a query sorting results by family tree generation and then by ORDPATH (putting siblings in order within a generation)

[xii] match (n:Person)-[z:Gedcom_DNA]->(m) with collect(m.RN) + collect(n.RN) as DM optional match path=(p:Person{RN:33454})<-[:father|mother*0..15]-(q:Person) where q.RN in DM with p,path,collect(last(nodes(path))) as cEnds optional match (q:Person)-[r:Gedcom_DNA]->(s:Person) where q in cEnds with r,p,[m in cEnds|m.fullname] as E with apoc.coll.dropDuplicateNeighbors(apoc.coll.sort(apoc.coll.flatten(collect (distinct E)))) as desc_tester match (m1:DNA_Match)-[rs:match_tg]->(t:tg) where rs.p in desc_tester and rs.m_anc_rn=33454 with t,apoc.coll.dropDuplicateNeighbors(apoc.coll.sort(apoc.coll.flatten(collect(distinct m1.fullname) + collect(distinct rs.m)))) as matches, apoc.coll.dropDuplicateNeighbors(apoc.coll.sort(apoc.coll.flatten(collect(distinct m1.RN) + collect(distinct rs.m_rn)))) as rns return t.tgid as tgid,t.chr as chr,t.strt_pos as strt_pos,t.end_pos as end_pos ,count(matches) as ct, matches, rns order by t.chr,t.strt_pos,t.end_pos

after which the person's count within their generation is computable (Table 3). This creates coordinates for graph visualizations: x=position in the generation and y=generation which would position a child under their parent and avoiding crossing descendancy lines. To this, we can then add a z-axis parameter such as a chromosome location.

The match_segment relationship is important in analytics and its subtleties merit further explanation. The relationship worksheet in Supplement 1 shows there are 371,653 match_segment relationships. Granular detail on tester and match links to segments is maintained by properties in the match_segment relationship. When querying using this relationship, it is important to recognize what is returned by cypher queries. These three queries look at this from the perspective of the author with record number of 1:

Query 1.
MATCH p=(m:DNA_Match{RN:1})-[r:match_segment]-() RETURN count(*)
Returns 133 Segments. A DNA tester will only show up in a DNA_Match node when they are a match in another tester's kit. While a tester may have many matches in their results, this query returns segments identified in other tester's results. The value of r.m_rn is the same as that of m.RN, so the query could be written differently with the same result: MATCH p=(m:DNA_Match)-[r:match_segment{m_rn:1}]-() RETURN count(*).

Query 2
MATCH p=(m:DNA_Match)-[r:match_segment{p_rn:1}]-() RETURN count(*)
Returns 6,490 match_segment relationships identified in the DNA tester's results. In this query there are multiple DNA_Matches. The match_segment relationship has a property p, for propositus (tester), which is the same as the DNA_Match node fullname. Thus, both the DNA_Match and match_segment returns are constrained. This query result represents the number of rows in the original FTDNA chromosome browser file.

Query 3
MATCH p=(m:DNA_Match)-[r:match_segment{p_rn:1}]-
(s:Segment) with collect(distinct s.Indx) as sc RETURN size(sc)
Returns 6,056 segments. The distinct in the collect function aggregates duplicates which, in this case, are 6490-6056 or 434, which is the number of times segments are shared by more than one match. It also illustrates how many segments are connected to only one match. This is illustrated further by assessing the degree of nodes.

Degree centrality is a central concept in graph theory. Degree is the number of relationships linking a node to others. Table 4 shows that less than 10% of DNA_Match nodes are linked to more than one segment. We can look at degree from the DNA match perspective. There are 340,784 matches with at least one Segment ≥ 7 cm and ≥500 snps. Of these 192,541 or 56% have a degree of 2 or more. The segment degree distribution is linear in a log-log plot (Figure 5) thereby conforming to the power law: $Pr(k) \propto k^{-\gamma}$ where k = degree and Pr(k) is the probability of k. The slope of the line is γ, in this case -3.3, which is expected with a scale free network[34].

FTDNA and GFG use different methods in computing matches. The GFG shared_match relationship is derived directly from the matches in a kit and retains FTDNA data as properties (see Appendix 6). The match_by_segment relationship is computed by aggregating match_segment data using its p and m properties

to assure fidelity to the original data and filters of cm ≥ 7 and snp_ct ≥ 500. The concordance of these two methods is computed by the UDF ***gen.quality.match_method_concordance*** (Table 5). The concordant set (86%) shows complete fidelity of the cm and snp data computed in the Stinnett project as shown by no rows returned in the second worksheet of the UDF report. This fidelity will not likely occur in endogamous scenarios where small segments are more prevalent and applied selection logic may differ with the two methods. Table 5 shows that the GFG match_by_segment methods produces a trivial number of matches not found by FTDNA; that is, very few false positives. In contrast, the shared_match produces more matches than match_by_segment. This discrepancy can be attributed to the confounded identities of matches which GFG does not presently manage because there are not unique identifiers for persons in the downloaded source files.

Table 2.. A deidentified example of 36 descendancy paths from a common ancestor [2] sorted in hierarchical order using an ORDPATH concatenated bitstring. The family tree record number (RN List) is a Dewey string which does not alphabetically sort in the hierarchical order of the family tree. UDF: **return gen.tree.descendancy_tree(2)**

ORDPATH	generation	RN List
12	1	[2]
12011	2	[2,1]
12011110078	3	[2,1,208]
0120111100781110049c0	4	[2,1,208,23320]
012011110078111004a65	4	[2,1,208,23485]
12011110079	3	[2,1,209]
012011110079111004ab3	4	[2,1,209,23563]
012011110079111004ab7	4	[2,1,209,23567]
01201111007911100534f	4	[2,1,209,25767]
01211007a	2	[2,210]
01211007a1101fd	3	[2,210,597]
01211007a1101fd11100541e	4	[2,210,597,25974]
01211007a1101fd11100541f	4	[2,210,597,25975]
01211007a1101fe	3	[2,210,598]
01211007c	2	[2,212]
01211007c11011d6	3	[2,212,814]
01211007c11013ec	3	[2,212,1348]
01211007c11013ec1101f5d	4	[2,212,1348,4277]
01211007c11013ec11100428a	4	[2,212,1348,21474]
01211007c11013ec11100428a1110054bb	5	[2,212,1348,21474,26131]
01211007c11013ec1110053f5	4	[2,212,1348,25933]
01211007c11013ec1110053f6	4	[2,212,1348,25934]
01211007c11013ed	3	[2,212,1349]
01211007c11013ed1110052d4	4	[2,212,1349,25644]
01211007c11013ed1110052e7	4	[2,212,1349,25663]
01211007c1101515	3	[2,212,1645]
01211007d	2	[2,213]
01211007d11011d1	3	[2,213,809]
01211007d11011d111100569a	4	[2,213,809,26610]
01211007d11011d1111005f5b	4	[2,213,809,28851]
01211007d1101a44	3	[2,213,2972]
01211007d1101a4411100541c	4	[2,213,2972,25972]
01211007d1101a4411100541d	4	[2,213,2972,25973]
01211007d1101a441110059e6	4	[2,213,2972,27454]
01211007d1101da5	3	[2,213,3837]
01211007d1101fbc	3	[2,213,4372]

Table 3. An example of 40 descendancy paths from a common ancestor [33454] sorted by generation and hierarchical order using ORDPATH so that the position within the generation is properly sorted.

Ancestor	gen	pos	path
Margaret Stinnett [33454] (1667-1727)	0	1	"33454>"
James Stinnett [27374] (1706-1777)	1	1	"33454>27374>"
Margaret Stinnett [33688] (1686-1729)	1	2	"33454>33688>"
Benjamin Hughes Stinnett [5427] (1710-1773)	1	3	"33454>5427>"
James Stinnett [29522] (1740-1831)	2	1	"33454>27374>29522>"
William Stinnett [29524] (1730-1795)	2	2	"33454>27374>29524>"
John Stinnett [29525] (1735-1795)	2	3	"33454>27374>29525>"
Rachel Elizabeth Stinnett [29526] (1740-1822)	2	4	"33454>27374>29526>"
William Benjamin Stinnett [5467] (1746-1831)	2	5	"33454>5427>5467>"
Priscilla Stinnett [5537] (1740-)	2	6	"33454>5427>5537>"
Susannah Stinnett [5541] (1750-)	2	7	"33454>5427>5541>"
Benjamin Mason Stinnett [5425] (1736-1799)	2	8	"33454>5427>5425>"
Dorcas Stinnett [5429] (1733-)	2	9	"33454>5427>5429>"
Jesse Stinnett [28870] (1774-1854)	3	1	"33454>27374>29522>28870>"
Riley Stinnett [29531] (-)	3	2	"33454>27374>29522>29531>"
John R Stinnett [5424] (1760-1831)	3	3	"33454>27374>29522>5424>"
Charles Stinnett [33506] (-)	3	4	"33454>27374>29525>33506>"
Elizbeth Flowers [33423] (1770-1846)	3	5	"33454>27374>29526>33423>"
Nancy Stinnett [5469] (1790-)	3	6	"33454>5427>5467>5469>"
Benjamin Stinnett [5470] (1783-)	3	7	"33454>5427>5467>5470>"
Reuben Stinnett [5471] (1774-)	3	8	"33454>5427>5467>5471>"
Lucy Stinnett [5513] (1772-)	3	9	"33454>5427>5467>5513>"
Joel Stinnett [5514] (1770-1847)	3	10	"33454>5427>5467>5514>"
Lindsey Stinnett [5516] (1783-1870)	3	11	"33454>5427>5467>5516>"
Richard Dewey Stennett [18251] (1790-)	3	12	"33454>5427>5467>18251>"
William Hightower Stennett [18706] (1788-)	3	13	"33454>5427>5467>18706>"
Elizabeth Stinnett [33413] (1776-)	3	14	"33454>5427>5467>33413>"
Benjamin Marion Stennett [19015] (1792-1876)	3	15	"33454>5427>5467>19015>"
William Stinnett [33657] (1771-1840)	3	16	"33454>5427>5467>33657>"
Charles A. Stinnett [5443] (1775-1853)	3	17	"33454>5427>5467>5443>"
Isham Stinnett [5543] (1769-1857)	3	18	"33454>5427>5425>5543>"
William S Stinnett [5557] (1761-1838)	3	19	"33454>5427>5425>5557>"
Benjamin Stennett [18833] (1761-)	3	20	"33454>5427>5425>18833>"
James Stinnett [33512] (1765-1850)	3	21	"33454>5427>5425>33512>"
John Stinnett [33513] (1771-1831)	3	22	"33454>5427>5425>33513>"
Bailey Stinnett [28868] (1828-1859)	4	1	"33454>27374>29522>28870>28868>"
Reuben Stinnett [29512] (1828-1912)	4	2	"33454>27374>29522>28870>29512>"
Thomas Stinnett [29516] (1805-1845)	4	3	"33454>27374>29522>28870>29516>"
James E Stinnett [29517] (1810-1860)	4	4	"33454>27374>29522>28870>29517>"

Table 4. The degree of the DNA_Match nodes is skewed. Over 90% of DNA_Match nodes only connect to one segment.

Degree	grp_ct	group total	%
1	335,791	335,791	90.35%
2	13,256	26,512	7.13%
3	1,662	4,986	1.34%
4	484	1,936	0.52%
5	165	825	0.22%
6	74	444	0.12%
7	44	308	0.08%
8	27	216	0.06%
9	19	171	0.05%
10	11	110	0.03%
11	17	187	0.05%
12	3	36	0.01%
13	3	39	0.01%
14	3	42	0.01%
15	2	30	0.01%
20	1	20	0.01%
		371,653	

Figure 5. The degree distribution is linear in a log-log plot. The linear regression r2=0.98; F-value=820; df = 14 and 1; p= 0.0000167; y = -3.29227X + 5.34895.

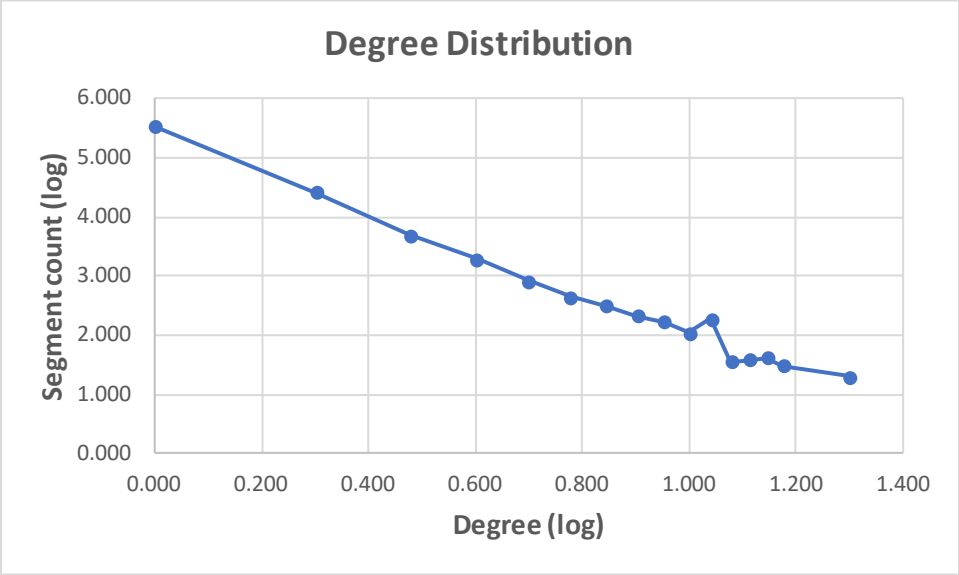


Table 5. Concordance between matching methods for the Stinnett project as shown in the first worksheet of the UDF output[xiii].

shared_match	match_by_segmen	both	ct		
Y	Y	Y	306,021		86.27%
Y	N	N	48,679		13.72%
N	Y	N	21		0.01%
			354,721		100.00%

The anc_desc property added to Segment nodes and the match_segment relationship enables queries limited to a very small subset of nodes and relationships associated with the common ancestor's descendants (Table 6). The seg_seq relationship spans a chromosome region defined by the triangulation group and closes the triangle between DNA_Match pairs who map to the region subsumed by a triangulation group. This strategy allows visualizing graph traversals from Kit and DNA_Match nodes without computing overlaps of the ordered segments. The triangulation group boundaries were not discrete enough to fully differentiate family branch members from others. We therefore explored the detail of the segments within the triangulation groups.

[xiii] UDF: *gen.quality.match_method_concordance()*

Table 6. Triangulation group segments associated with descendants of a common ancestor. A small subset of all segments in a triangulation group actually map from the descendants.

name	cm	chr	strt_pos	end_pos	seg_ct	seg_in_seq	%
01-001-011	19.6	1.	1,941,929	11,375,944	469	16	3.41%
01-054-067	18.8	1.	54,689,461	67,560,097	753	22	2.92%
01-234-239	10.5	1.	234,636,595	239,164,627	245	18	7.35%
02-079-101	12.1	2.	79,855,622	101,612,039	1,347	16	1.19%
02-120-135	16.5	2.	120,986,872	135,211,667	629	10	1.59%
02-187-200	9.2	2.	187,621,516	200,647,533	1,481	16	1.08%
04-054-108	46.9	4.	54,762,082	108,528,923	1,523	38	2.50%
04-118-131	9.7	4.	118,330,973	131,468,686	567	18	3.17%
05-038-064	14.1	5.	38,780,436	64,423,143	294	40	13.61%
05-148-169	27.1	5.	148,824,841	169,862,468	544	20	3.68%
06-023-033	8.4	6.	23,437,273	33,945,921	3,551	14	0.39%
06-094-134	35.4	6.	94,222,638	134,090,307	794	20	2.52%
07-012-021	15.3	7.	12,081,336	21,563,636	250	12	4.80%
07-110-139	26.4	7.	110,088,623	139,939,855	294	18	6.12%
09-000-012	26.1	9.	46,587	12,316,830	395	36	9.11%
09-038-088	24.2	9.	38,736,897	88,104,161	2,074	36	1.74%
10-055-076	20.7	10.	55,168,137	76,814,527	562	30	5.34%
12-118-129	21.8	12.	118,274,349	129,178,435	316	38	12.03%
15-047-068	23.0	15.	47,994,703	68,117,156	705	14	1.99%
15-089-102	34.4	15.	89,595,883	102,428,887	337	30	8.90%
19-035-059	46.9	19.	35,843,545	59,097,160	405	34	8.40%
20-039-062	49.9	20.	39,968,188	62,948,788	673	26	3.86%
22-047-051	12.9	22.	47,005,952	51,150,473	195		
					18,403	522	2.84%

On initial inspection, triangulation groups are considerably more complex because in a multi-kit environment there are numerous continuously overlapping segments. Remarkably, with numerous kits graph traversals discover a very small subset of specifically discernible segments associated with descendants sharing a common ancestor. The distinctive feature of these segments is their boundaries, which represent crossover points. Table 6 summarizes some triangulation groups associated with common ancestors. Triangulation groups contain numerous segments, but on average only 2.6% are attributable to the common ancestor's descendants.

Graph traversal queries on the Stinnett project graph discovered 1618 segments with two known testers who shared a common ancestor at various positions in the Stinnett family tree. To distinguish them, they are called ancestor-associated segments (AAS). These AAS segments overlap with other segments that are not linked to the common ancestors in these initial discovery queries. When these AAS segments were used to query for matches from the total population, which has over 350,000 segments in the Stinnett project, only 1696 new match pairs were found. That is, 78 new match pairs, who did not yet have a known common ancestor were found. This indicated that AAS segments were relatively specific for the common ancestor.

Next, monophyletic segment set (MSS) supernodes were created to subsume the small subset of AAS segments associated with a common ancestor[35]. MSS node properties were the ancestor's record number (mrca) and fullname. A MSS-seg relationship connects the MSS node to the subsumed AAS segments, enabling an efficient

query returning their ancestor specific segments (UDF: ***gen.report.monophylytic_segment_set_report)***. To facilitate queries, a new property was added to these AAS Segment nodes to identify them as associated with the most distant common ancestor. These segments represented, on average, only 0.53% of all chromosome segments (Table 7)[xiv]. MSS nodes associated with common ancestors may have DNA matches from another branch of the family tree who might be identified by a distinct segment (Figure 6)[xv].

Table 7. Monophyletic segments associated with descendants of a common ancestor.

chr	chr_total_segs	mss_seg_ct	percent
1	31,580	140	0.44
2	30,242	109	0.36
3	20,488	93	0.45
4	22,830	77	0.34
5	19,586	105	0.54
6	21,853	105	0.48
7	18,122	69	0.38
8	15,358	93	0.61
9	17,873	99	0.55
0X	22,391	65	0.29
10	17,331	108	0.62
11	13,969	102	0.73
12	18,089	90	0.5
13	10,286	57	0.55
14	9,791	61	0.62
15	11,888	51	0.43
16	11,617	87	0.75
17	10,649	76	0.71
18	8,510	69	0.81
19	5,572	57	1.02
20	8,602	81	0.94
21	4,891	53	1.08
22	5,259	43	0.82
	356,777	1,890	0.53

In some scenarios you can determine where in the family tree the crossovers occur. DNA Painter rendering visualize the MSS segments (Supplement 3). The intermediary MRCA (iMRCA) share the most distant common ancestor (MDCA). Recognizing this in reports requires sorting the MSS by the position of their monophyletic ancestor in the family tree using ORDPATH (Table 8).

[xiv] MATCH (m:MSS)-[r:ms_seg]->(s:Segment) with m,left(s.Indx,2) as chr with chr,count(*) as ct optional match (s2:Segment) where s2.chr=chr with chr,ct,collect(distinct s2.Indx) as sc with chr,size(sc) as chr_total_segs,ct as mss_seg_ct return chr,chr_total_segs,mss_seg_ct,apoc.math.round((toFloat(mss_seg_ct)/chr_total_segs)*100,2) as percent order by chr

[xv] MATCH p=(mss:MSS{mrca:63})-[r:ms_seg]->(s:Segment)-[rm:match_segment]-(m:DNA_Match) return p

Figure 6. The MSS of Samuel Lewis Stinnett, Jr, born in 1822 in Tennessee, has 10 AAS segments linked to 8 putative descendants, 6 of whom are positioned in the family tree and show a record number. Two other matches have circumstantial evidence they are descendants and their sharing on a segment with two others known to be in the family tree supports this hypothesis.

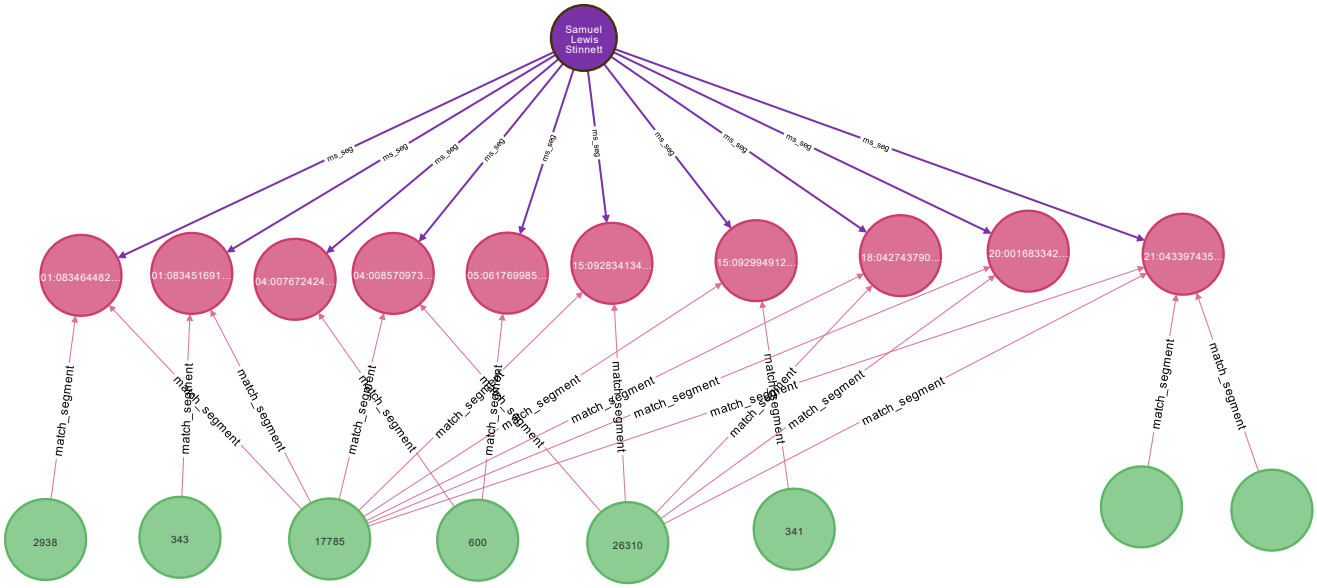


Table 8. Selected segments showing its MSS mapping to 3 different monophyletic ancestors who are descendants of a single more distant common ancestor [5467].

mrca	seg	gen	descendant path rns
5443	01:000752566:004913940	4	[33454,5427,**5467**,5443]
29008	01:000752566:004913940	7	[33454,5427,**5467**,5443,5436,5609,29008]
29008	01:000752566:004913940	7	[33454,5427,**5467**,5443,29011,5608,29008]
5467	01:159700039:193826947	3	[33454,5427,**5467**]
18256	01:159700039:193826947	6	[33454,5427,**5467**,5516,5529,18256]

GFG uploads Y-DNA data from the FTDNA Family Finder and Y-DNA csv files and from the FTDNA haplotree. This allows a visualization of matches aligned with their position on the Y-haplotree (Figure 7).The UDF ***gen.tree.patrilineal_lineage*** creates both a patrilineal tree (Table 9) and an ordered list of all patrilineal men in the family tree (Table 10). The UDFs ***gen.tree.matrilineal_lineage*** and ***gen.tree.x_chr_lineage*** create similar matrilineal and X-chromosome inheritance trees.

Figure 7. Y-haplotree (75 red nodes) with Y-DNA testers (38 green nodes) positioned on the tree. They are connected by 38 match_block and 74 block_child relationships. Men who have the same haplogroup may appear on nearby nodes if their testing involved an incomplete set of SNPs. This typically results from newly discovered blocks and the lack of that knowledge when earlier tests were performed. The visualization can be viewed using the svg file in Supplement 4. The testers are anonymized by showing only their node id.

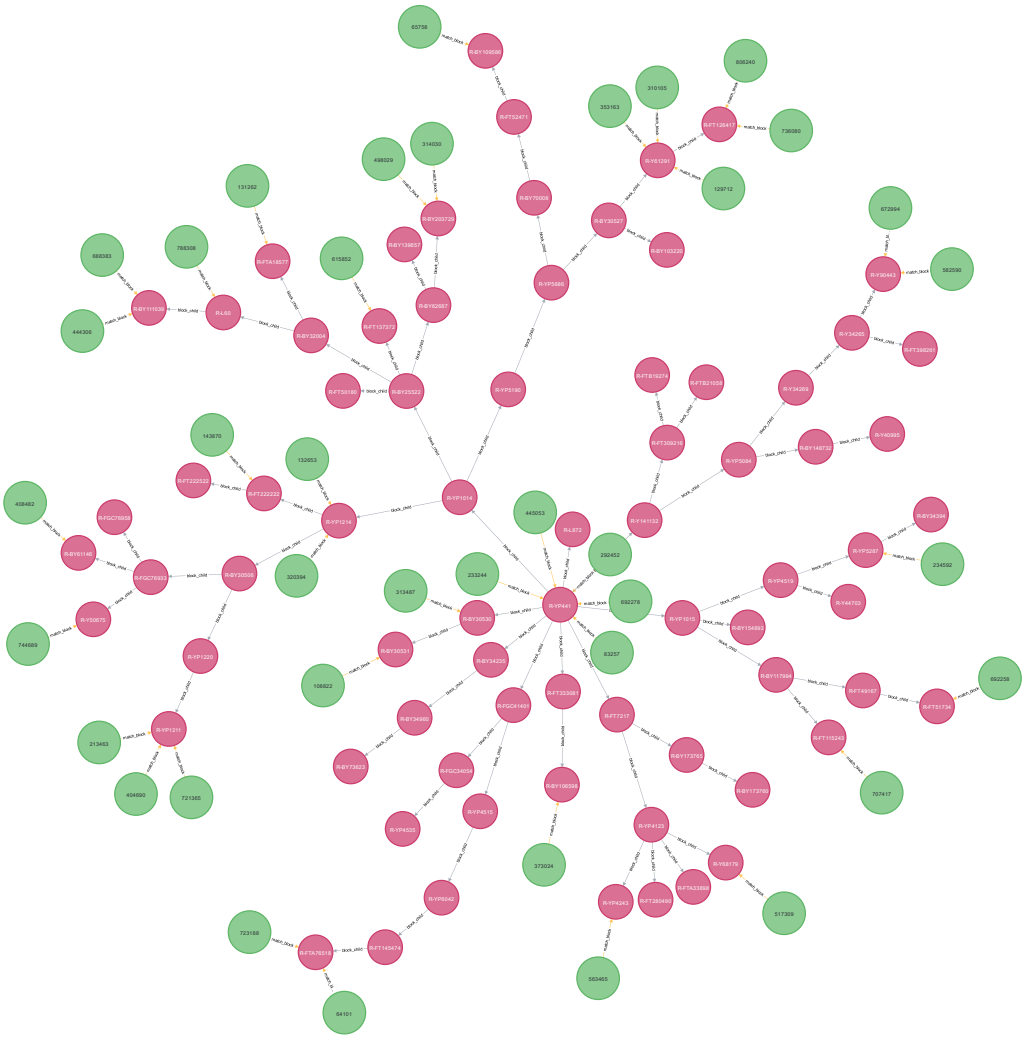


Table 9. Patrilineal ascendancy tree which includes a fictitious MRCA to assist in linking lines hypothesized to share a common ancestor.

Name	gen	Ahnentafel
Stinnett Male MRCA [27085] (-)	11	2048
William Richard Stinnett [25813] (1645-1699)	10	1024
William Stinnett [25815] (1682-1735)	9	512
James Stinnett [27374] (1706-1777)	8	256
James Stinnett [29522] (1740-1831)	7	128
John R Stinnett [5424] (1760-1831)	6	64
Samuel Stinnett [100] (1784-1850)	5	32
Samuel Lewis Stinnett [63] (1822-1864)	4	16
Samuel Henry Stinnett [679] (1854-1928)	3	8
anonymized	2	4
anonymized	1	2
anonymized		1

The GFG-PI UDF **gen.algo.triangle_count** returns a list of DNA matches with the count of triangles each match forms with their matches[xvi]. The sum of these numbers is not the number of triangles because of the permutations possible. Triangles are composed of three matches and the combination observed is the number of triangles. In the Stinnett project, 14 of the 65 kits form 24 shared_match triangles with matches who are in the GEDCOM (Table 11). When all matches are considered, there are 5377 shared match triangles involving 2736 matches. The match_by_segment triangles involve fewer matches. There are three reasons for this: 1) some matches in GEDCOM section are attributable to the confounded matches with are excluded from match_by_segment (see Table 5); 2) match_by_segment is computed using only matches in the GEDCOM whereas the shared matches reported by FTDNA do not have this constraint; and 3) the project kits were selected because they had a Stinnett ancestor, making matches with other branches of the family tree much less likely. Triangles are a starting point for segment triangulation.

The criteria for robust segment triangulations[36] are:

1. A triad of matches: A, B, C, as just discussed
2. Shared matches for each possibility: A-B, A-C and B-C (requires 3 DNA testers)
3. The match pairs share overlapping segments
4. The match pairs share a common ancestor. Optional while discovering potential new descendants.

The UDF *gen.discovery.triangulated_segment_matches* discovers all possible triangulations meeting these criteria. The report returns a list of triads with their common ancestors and triangulated segments. It also reports each segment and all the matches triangulating to it. The visualization query in the report can run in either GraphXR (Figure 8), the Neo4j browser (Figure 9) or with DNA Painter (Figure 10).

[xvi] UDF example is *gen.algo.triangle_count(2,7,3500,false)* where the parameters are match type (1=shared_match; 2=match_by_segment), minumum cm, maximum cm, and whether to limit query to known matches or all matches.

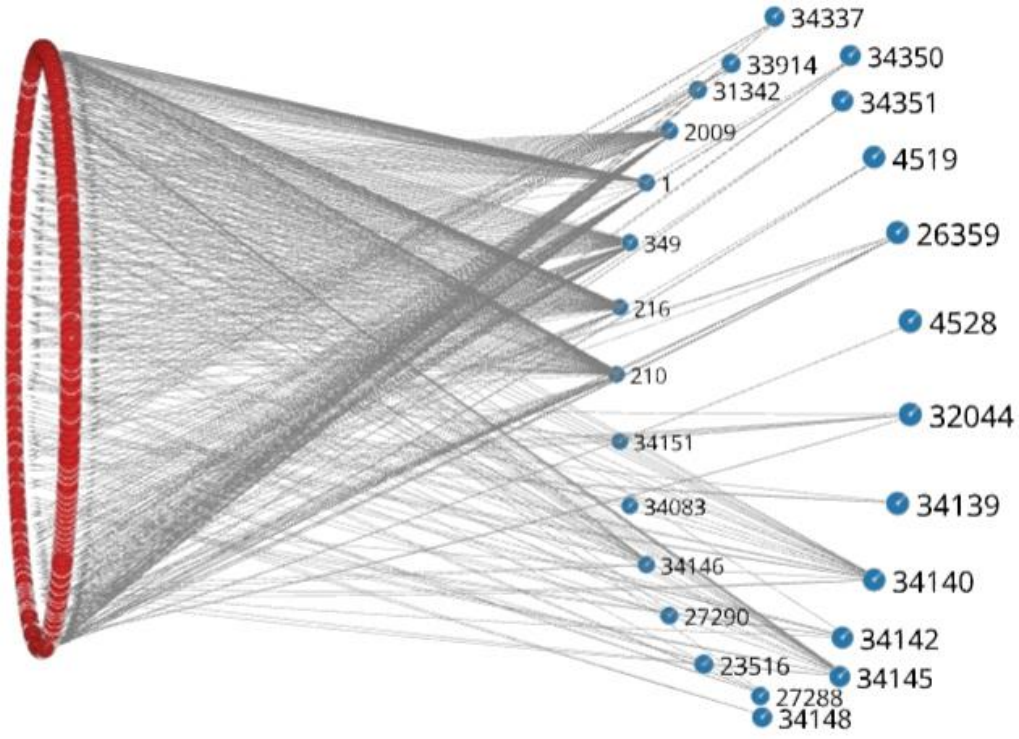
Table 10. Patrilineal relatives shown in a descendancy tree from the most distant patrilineal ancestor. The report sorts by ORDPATH and generation using the latter to indent the names for easier interpretation of the descendancy. When available, the Y-haplogroup is displayed. More recent men are anonymized. Only the top 41 rows (of 706) are shown.

Fullname	gen	YHG
Stinnett Male MRCA 27085(-)		
William Richard Stinnett 25813(1645-1699)	1	
...William Stinnett 25815(1682-1735)	2	
......James Stinnett 27374(1706-1777)	3	
.........James Stinnett 29522(1740-1831)	4	
...........John R Stinnett 5424(1760-1831)	5	
.............Samuel Stinnett 100(1784-1850)	6	
...............Samuel Lewis Stinnett 63(1822-1864)	7	
.................James Samuel Stinnett 675(1842-1915)	8	
.....................William Stinnett 1777(1873-)	9	
.....................James Monroe Stinnett 1778(1875-1963)	9	
.....................George Wesley Stinnett 1779(1877-)	9	
.....................John Calvin Stinnett 1781(1881-1949)	9	
.....................Albert Nathaniel Stinnett 1782(1887-1920)	9	
.................Rufus Calvin Stinnett 676(1845-1890)	8	
.....................William Calvin Stinnett 1783(1872-1956)	9	
.....................William Clyde Stinnett 1785(1900-1973)	10	
	11	
	12	
	11	
.....................Marvin Urriah Stinnett 1790(1906-)	10	
.....................William Franklin Stinnett 1792(1932-1932)	11	
	11	
.....................Samuel Lewis Stinnett 1797(1880-1918)	9	
.....................Rufus Jessie Stinnett 1800(1886-1887)	9	
.................Thomas Nathaniel Stinnett 678(1850-1928)	8	
.....................Sam M. Stinnett 1810(1879-1879)	9	
.....................John Nathaniel Stinnett 1811(1881-1942)	9	
.....................John Mryle Stinnett 27319(1903-1981)	10	
	11	
	12	
.....................George Uriah Stinnett 1812(1883-1955)	9	
.....................Charlie E. Stinnett 1813(1885-)	9	
.....................Sidney Stinnett 27023(-)	10	
.....................Dale Stinnett 27025(-)	11	
.....................Thomas Nathaniel Stinnett 29560(1906-1976)	10	
.....................Rufus Calvin Stinnett 1820(1898-1921)	9	
.................Samuel Henry Stinnett 679(1854-1928)	8	
.....................Marion Marshall Stinnett 1081(1897-)	9	
	10	
	11	I-BY64293

Table 11. Triangle algorithm results using four different variations of parameters. The percent row is that of the match_by_segment row compared to the shared_match row.

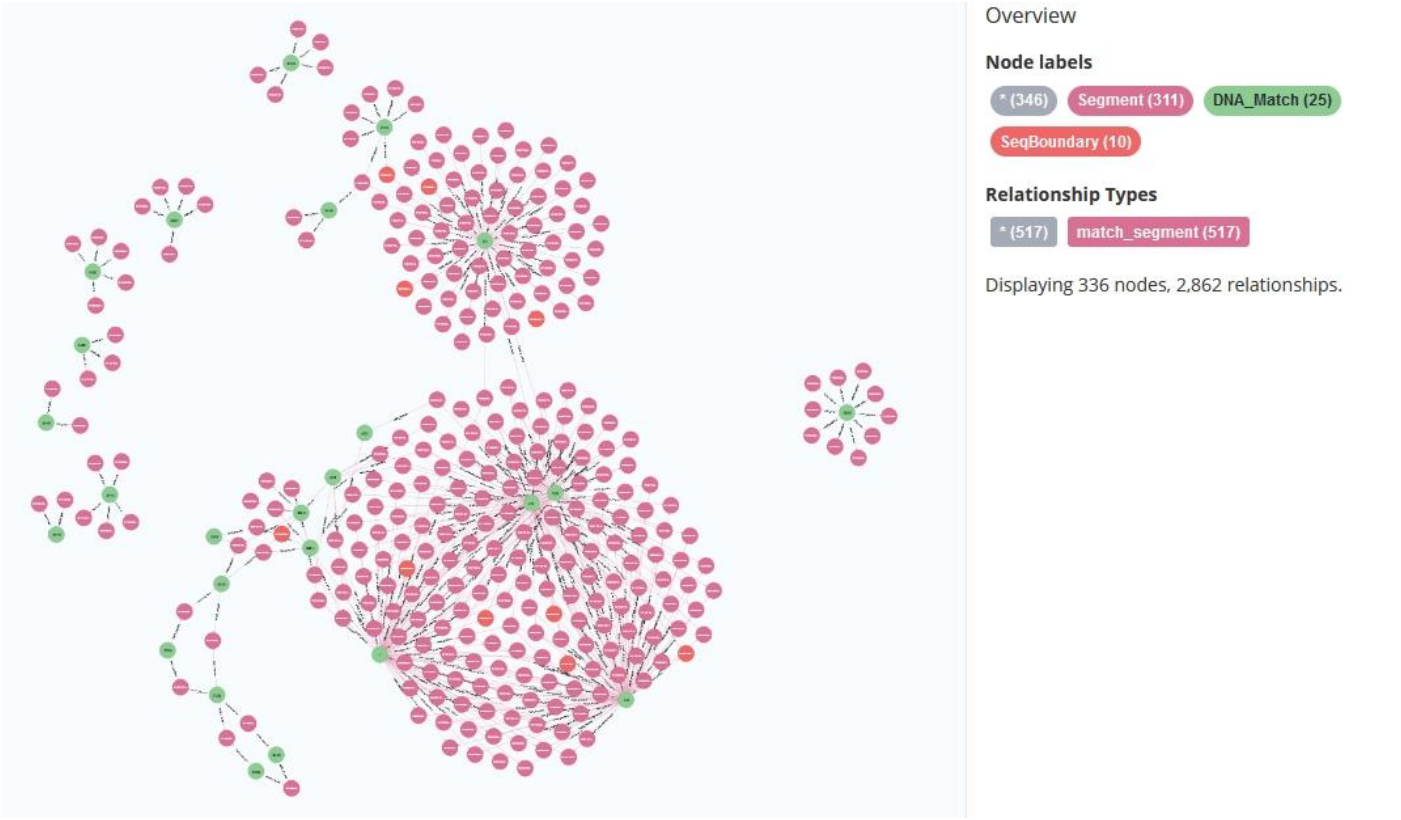
	matches in GEDCOM			all matches	
	matches	**triangles**		**matches**	**triangles**
shared matches	11	24		2736	5377
match by segment	9	20		1023	5300
percent	82%	83%		37%	99%

Figure 8. GraphXR rendering on DNA_Match nodes (blue) mapped to triangulated segments. Individual segments may not map to three DNA_Match nodes because the triangulation is based on overlapping segments.



Degree is the number of relationships for a node. A DNA match with a degree of one offers no direct useful genealogical information because it does not link other matches together. It participates in no triangles. The UDF **gen.algo.degree_centrality** computes the degree for each node. The UDF **gen.algo.prune_uninformative_matches** re-labels DNA_Match nodes. For the Stinnett project with its original 250,520 DNA_Match nodes, the UDF labels 74,707 nodes with degree > 1.

Figure 9. Neo4j Browser rendering of DNA_Match nodes (green) mapped to triangulated segments (red). Individual segments may not map to three DNA_Match nodes because the triangulation is based on overlapping segments.

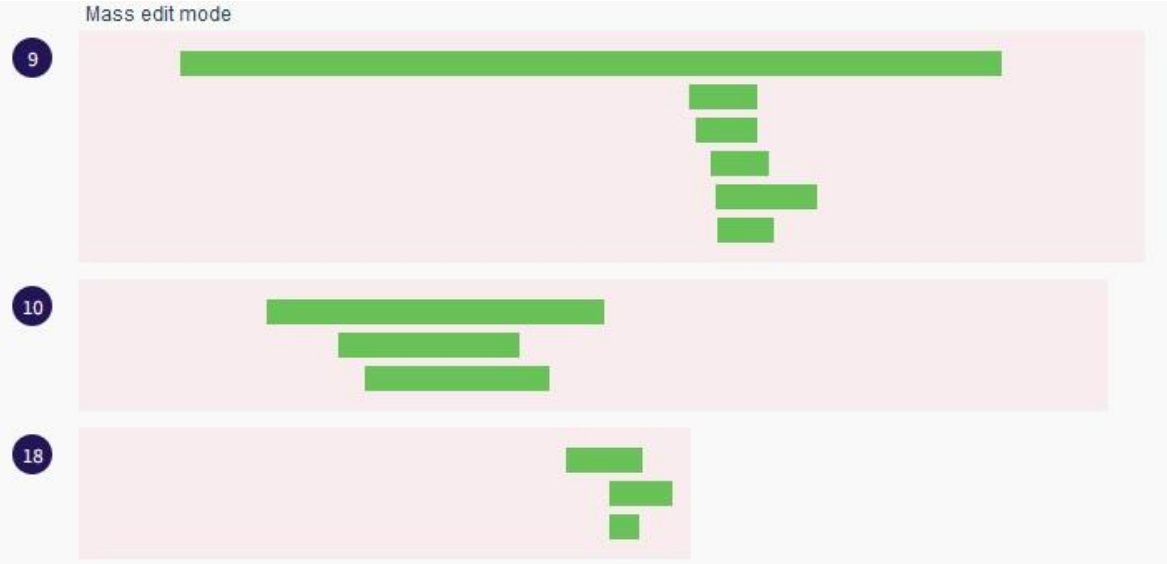


The GFG-PI supports some updates of the existing database. To update the triangulation group curated file use **load_tgs_from_template. load_tgs_from_csv.** Users can re-run the enhancements from GSGS but will then need to re-run **gen.tgs.setup_tg_environment** because the data it creates is erased to accommodate the new triangulation groups.

The UDFs **gen.algo.community_detection_icw** and **gen.algo.community_detection_shared_matches** respectively use match_by_segment and shared_match relationship to create communities. Each has three options for clustering algorithms from the Neo4j GDS plugin: Louvain, modularity optimization and page

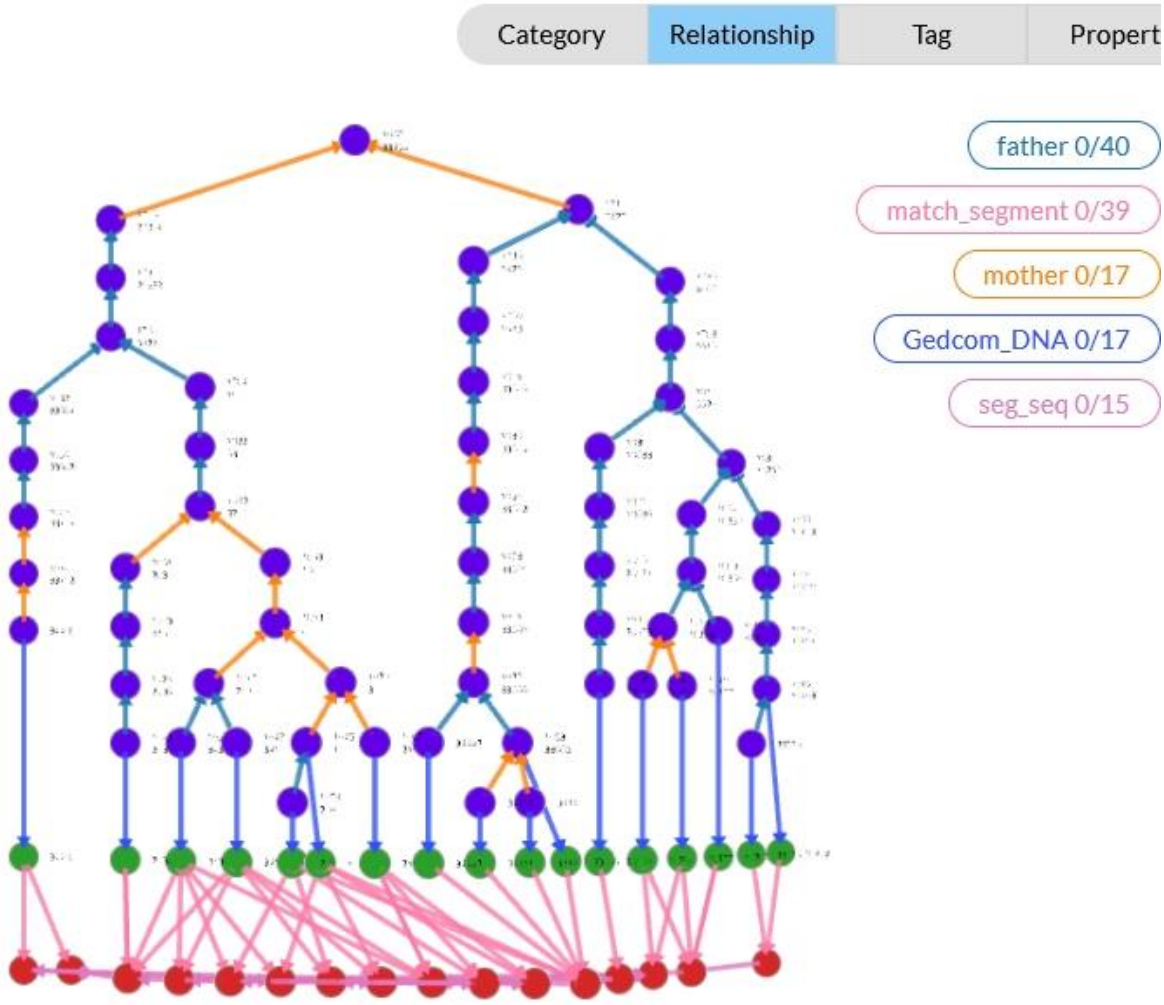
There are some analytical challenges. Identity of matches is sometimes confounded because of ambiguous names in the original data. These confounded matches are identified using the UDF **gen.quality.confounded_matches**. Their distinct identity is evident because they have multiple rows in the FTDNA Family Finder csv files and the rows have distinguishing shared DNA values. When this UDF is executed, a new property confounded='Y' is added to the identified DNA_Match nodes. Caution is needed when these matches emerge in analytics.

Figure 10. Sample of triangulated segments shown in DNA Painter. The matches share a common ancestor, the authors great great grandfather.



Individual triangulation groups can be visualized in GraphXR ().

Figure 11. Individual triangulation group in the Stinnett project shown person (purple), DNA matches (green and segments (red).



Discussion

Graph methods are well suited for genealogy data management and analytics**.** Presently there are very few graph tools in use today by genealogists. The GFG-PI and GFGS can encourage adoption of graph methods prior to mastering the underlying methods. Genealogists should see immediate benefit in their research using GFG UDF functions. The queries in reports expose can be further refined by adjusting parameters using logic familiar to experienced genealogists.

The metrics for assessing the benefits of graph methods in genealogy are important. Adoption of these methods will require demonstrated and measurable improvements over extant methods. The CODASYL Data Model, initially developed in 1959, laid out optimal requirements for data management and resulted in development of relational databases. Basic requirements were keys, scans and links. With the emergence of big data in the 1990's, No-SQL databases emerged, eliminating tables and using scanning methods to find relevant key:value data. No-SQL also uses scale up systems deploying multiple servers. These earlier systems exceled in their use of keys and scanning. Graph databases were motivated by the shift in thinking from optimizing data management to optimizing data retrieval. Graph databases accomplish this using traversals rather than joins (relational systems) or scans (No_SQL). Traversals along graph edges (called relationships in Neo4j) transformed the ability to link different sets of data needed to solve complex questions. The Neo4j architecture links several discrete graphs (Table 12), enabling traversals from an ancestor through his or her descendants (Person nodes) to the associated DNA tester (DNA_Match nodes) to their chromosome segments (Segment nodes) and then to other matches.

"Graph thinking" plays a key role in realizing the value of graph methods[37]. Genealogists see many individual component graphs that are related, interact, and influence each other. Rather than considering them separately, graph methods integrate them into a usable commodity. Specifically, we can use four main patterns to advantage: neighborhoods, hierarchies, paths, and recommendations. GFG is designed to make recommendations for further research. Thus, the ultimate measure of success will be the value genealogists attach to the insights GFG provides.

Table 12. GFG incorporates multiple graphs highlighted in green and orange. Orange contains definitions for columns and rows with the same name. Green indicates graphs linked in the GFG graph schema.

Graph name	Family tree	DNA matches	Geography	Chromosome map	Haplotree
family tree	family relationships				
DNA matches	GEDCOM RN to kit number	shared DNA			
Georgraphy	Event place lat-long	phylogeographic maps	adjacent geo-locations		
Chromosome map	DNA match to chr position	match segments	population specific segments	Segement sequence	
Haplotree	haplogroup to haplotree	tester's haplogroup	phylogenetic maps	haplogroup to chr region	Sequential mutations

GFG recommendations consider genealogy workflows by identifying the most relevant matches for future research and providing context such as their membership in triangulation groups, communities (family tree branch), their ancestral surnames, or the sources where more details may emerge. The GFG-PI uses several community detection algorithms to group individuals and align them with branches of the family tree. Reports include DNA matches, putative common ancestors, and chromosome segments which can be seamlessly uploaded to DNA Painter. Hierarchical clustering uses similarity matrices and iteratively identifies communities of similar nodes. There are two strategies. Agglomerative clustering merges nodes with similarity into clusters. Divisive clustering separates low similarity nodes from communities[34]. GFG-PI uses the latter, specifically degree-centrality, to identify and tag less relevant nodes. The Louvain algorithm agglomerative clustering produces intermediary clusters, forming a dendrogram, that conforms to family tree branches. To implement these community detection methods, GFG-PI uses Neo4j's Graph Data Science plugin[38]. GraphXR provides its own community detection algorithms.

GFG-PI uses several strategies to optimize the Neo4j graph schema and thereby increase query efficiency (Table 13). Indices of node properties speed the trip to the starting node(s) of traversals. Indices also accelerate loading the database, so they are created before the loading begins. You do not need the properties in the database to create an index. Query tuning involves limiting the number of nodes at the start of traversals.

Table 13. Strategies used to enhance GFG-PI performance.

Enhancement	benefit
indicies	fast finds where traversals start
specific query start points	limits the nodes initiating traversals
query constraints	limit paths traversed
redunant properties	reduce traversal lenghts
tag confounders	exclude elements from queries
close triangles with new relationships	one-time vs recurrent computations
standoff properties	reduce overstap computations
user defined function	operate closer to Neo4j server
pruning	removes non-informative elements

Constraining traversals will greatly decrease execution time. For instance, adding a simple constraint (***Query 4***) to a typical common ancestor query (***Query 5***) reduces the query time from about 2 seconds to ~10 milliseconds. This efficiency allows GFG to generously use common ancestors in reports which were previously avoided because of long computation times.

Query 4. match (p1:Person{RN:1})-[r1:father|mother*0..15]->(mrca:Person)<-[r2:father|mother*0..15] -(p2:Person{RN:600}) **where p1.RN<p2.RN** return collect(mrca.fullname) as mrca

Query 5. match (p1:Person{RN:1})-[r1:father|mother*0..15]->(mrca:Person)<-[r2:father|mother*0..15]-(p2:Person{RN:600}) return collect(mrca.fullname) as mrca

A redundant property allows a query to bypass a node during a traversal. For instance, the DNA_Match nodes initially only have data provided by FTDNA. By adding the GEDCOM record number to these nodes allows queries to start at the DNA_Match node rather than the Person node from which the record number was obtained. Similarly, by adding the GEDCOM fullname to the DNA_Match node, you can include it in the report without looking it up repeatedly in the Lookup or Person node.

Some nodes are not relevant to genealogy analytics. They are loaded because their limited value is only revealed when multiple testers are available. Analytics reveals, for instance, that their degree=1 means they are not informative because they do not link to multiple matches or form triangles. These nodes can either be "tagged," pruned or a hybrid solution. The DNA_Match nodes with a degree=1 can be relabeled, which makes them unrecognizable to queries. But they can be restored by reverting to the original label. However, seemingly uninformative nodes may be subsumed by supernodes (see below), making them useful.

Closing triangles enables graph algorithms. The shared_match and match_by_segment are examples. A special type of triangular closure is a so-called "standoff" property[39]. GFG-PI adds the p, m, cm and other properties to the match_by_segment relationship. These properties are derived from the two nodes linked by the relationship. Because Neo4j relationships are now indexable, they can be the start search target rather than looking at each node. These relationships are sometimes also described as "shortcut edges" because they bypass unnecessary steps.

Longer running queries can slow down processing, in part because Java garbage collection can be problematic. Neo4j has addressed this through batching in LOAD CVS and APOC periodic commit. APOC also provides UDF with recovery options that manage server interrupts during query execution. Nonetheless, one of the issues with Neo4j is that it was built with Java, which is not as efficient in managing garbage collection as other coding languages. User defined functions (UDF) were specifically developed by Neo4j to extend its capabilities and flexibility. They are written in java, the language behind Neo4j. They thus interact directly with the Neo4j server rather than through other internal interfaces. The same query runs many times faster in a UDF than through the Neo4j Desktop.

Degree is a simple and safe way to prune a genealogy graph. If a node has a degree of 1 it connects to only one other node and is not directly informative in genealogy analytics. Triangles have specific roles in graph thinking and analytics. Over 90% of segments and 40% of matches have a degree of one and would be useless without the creation of virtual graphs or supernodes such as triangulation groups and monophyletic segment sets. Figure 11 illustrates triangulation from a graph thinking perspective.

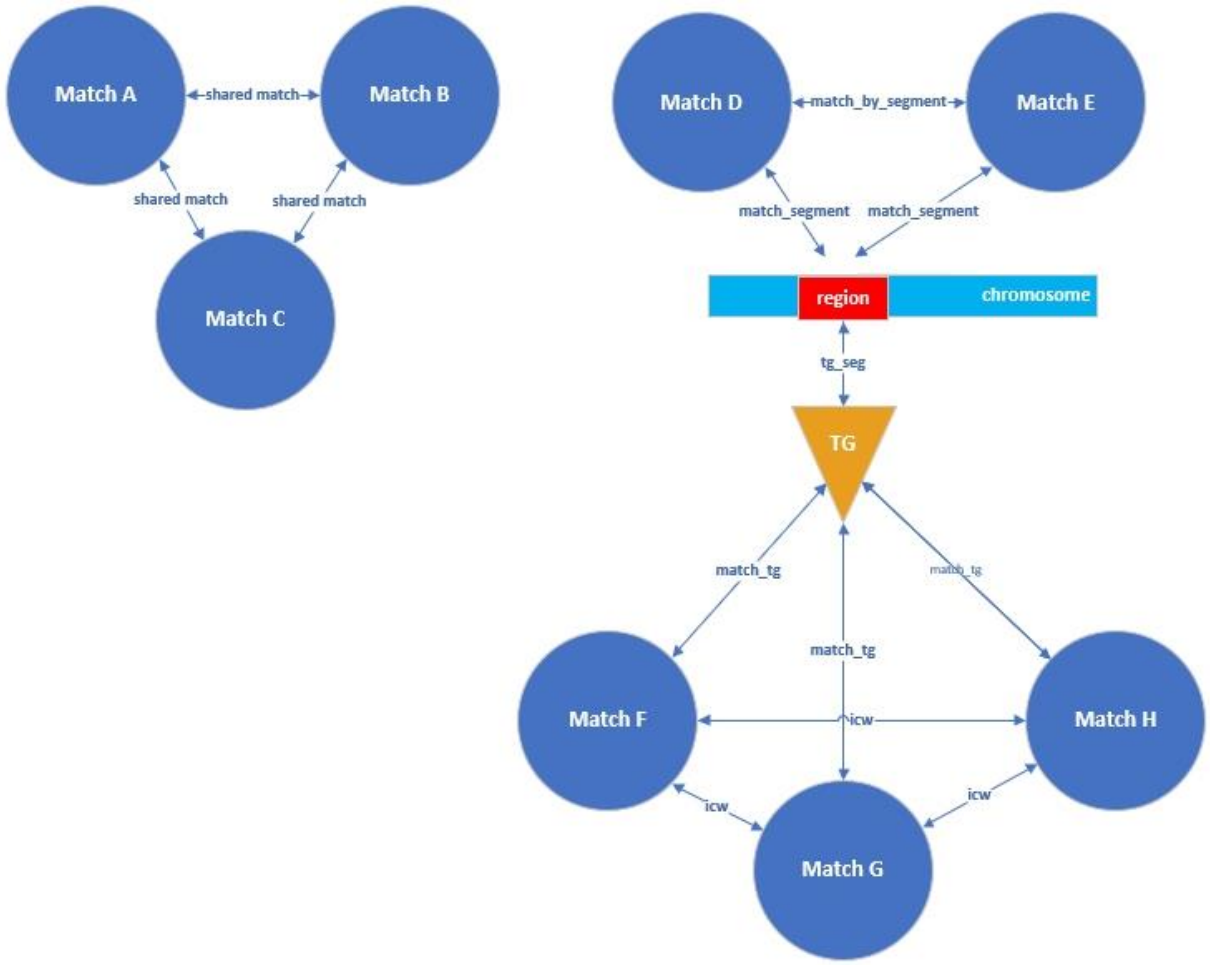
The term triangulation group is ambiguous. GFG uses several types of triangulation grouping, which need specific definition and distinguishing names (see Figure 11). DNA matches are a pair of individuals sharing DNA segments. If we have three match pairs that contain 3 nodes whose combinations (A-B, A-C and B-C) form a triangle, we can call these match triangulation groups (mTG). The mTG match pairs may not share a common

segment or ancestor. Genealogy triangulation groups (gTG) are three or more DNA matches with a known common ancestor who map to the same chromosome region.[40] Algorithmic triangulation groups (aTG) are formed by a match pair who share a segment (A->segment<-B). Graph algorithms use aTG to assemble triangles and create communities. Triangles have a central role in graph theory and analytics. If two triangles with 4 distinct individuals share a common edge then there are two individuals not on the shared edge who are nearest neighbors. In genealogy we may have an individual with 10 triangles, some with and some without common edges with other triangles. The structure or topology of such graphs form communities or clusters.

Different relatives whose test results are in the match list have overlapping segments mapping to the region defining the genealogy triangulation group (gTG). The borders of the gTG are fuzzy because one boundary can be outside those used to define the gTG. There is no precise way to resolve this fuzziness. Another ancestor on the same side of the family may have an adjacent gTG. GFGS uses DNA Painter[27] to produce chromosome maps with identified regions associated with ancestors.

The complexity increases dramatically when multiple kits are used. Graph thinking leads to discovery of remarkably simple and useful insights. Graph traversals from ancestors on different branches of the family tree via DNA matches terminate at very few segments. This graph is like a phylogeny tree with a common ancestor suggesting the name monophyletic segment sets (MSS). MSS nodes were created with properties of mrca (ancestor record number) and fullname (of the ancestor). When there is more than one MRCA a MSS node is created for each. A monophyletic ancestral couple can be constrained, selecting one person in the couple, if there is a more distant monophyletic ancestor sharing MSS segments with only one of the couple's MSS.

Figure 12. Triangulation group types. Match A, B, and C form a triangle when there are shared matches for A-B, A-C and B-C, as shown (mTG). Neo4j Graph Data Science creates a virtual graph using only Match D and E and the centimorgan-weighed match_segment relationship to algorithmically create triangles (aTG) and communities. Graph thinking views a robust triangulation group (gTG) as a supernode for mutually in common with (icw) Match F, G and H with properties for the chromosome region and MRCA shared by Match F, G, and H.



Graph thinking was revolutionized by Barabási who promoted the concept that networks self-organize because some nodes were preferred and form scale free networks featuring supernodes. Misteili, commenting on recent epigenetic research observes: "An important realization from these studies has been that the organization of genomes is characterized by a high degree of order and non-randomness." [41] Self-organization is truly remarkable in early embryos who unwrap and reconfigure chromosomes independent of parental DNA. Supernodes and hierarchical organization are also now identified for epigenetic elements which might represent Goldilocks segments for genealogists.

Scale free networks do not behave as random systems following normal distribution statistics. The Barabási-Albert model[34] based on preferred attachments conforms to the power law, characterized by a long tail. It is notable that such a pattern is observed in the shared centimorgan data at more distant relationships[19]. Figure 5 shows GFG segment degree distribution consistent with the power law. However, caution is necessary because there are alternative explanations for conformance or pseudo-conformance to the power law[42],[43]. The long tail represents high degree nodes, whereas the rare MSS nodes are low degree and buried within a large group of other low degree nodes where we expect nodes attributable to distant ancestors. The computational challenge is finding MSS segments. Graph methods provide a robust solution to this problem when there is enough data. The paradox of this "big data" solution is that we are seeking the small data pointing to one ancestor. Graph methods are particularly adept at such discovery.

There are important limitations to monophyletic segments. The start and end positions of segments are imputed, and the algorithms vary between vendors[44]. For this reason, this study was limited to FTDNA test results. FTDNA does not utilize family tree data in its matching algorithms[45]. Yet with graph methods, monophyletic segments were identifiable. Prior work by Bartlett[46],[40] shows examples of specific segment boundaries repeated within triangulation groups. Other genetic genealogists have reported finding segments attributable to distant ancestors. The present study incorporates numerous DNA matches; up to 250,000 in one project. This is important because the odds of finding matches sharing a distant ancestor are increased[47]. This opportunity is further augmented when, as in the present work, the matches are to a group of testers selected because they share a common ancestor.

Comparing and contrasting features of triangulation groups (gTG) and MSS is informative. Overlaps are irrelevant in identifying MSS segments and challenging for gTG. The boundaries in MSS are discrete and fuzzy for gTG. Some MSS precisely align with a few population specific segments used by FTDNA in population grouping. MSS segments could be used to create new population markers if the monophyletic ancestor's demographics are known.

MSS form an autosomal haplotree. While individual MSS segments are a rarity, the pattern of sets of segments is also rare. MSS supernodes connect to multiple segments on different chromosomes. These patterns can identify matches associated with the family tree branch and rare unknown matches who are candidates for further research. Boundaries of specific segments in both gTGs and MSS represent crossover points. Because these will vary on different descendancy branches they may prove useful in delineating the branch. This could be particularly helpful in endogamous families by enabling separate tracking of segments through different paths in the family tree. Goldilocks markers would have utility in forming more robust haplotrees akin to Y- and mt-haplotrees.

There are multiple overlapping segments in genetic genealogy datasets, particularly notable when using numerous kits. A triangulation group contains many overlapping segments because of variance in different match-pairs segments. It appears self-evident that the specific defining boundaries of MSS may be obscured within the triangulation group region. However, graph traversals from the ancestor to the segment enables their discovery. What distinguishes monophyletic segments from others is their boundaries which represent pseudo

crossover points (PCOP); pseudo because the boundaries are approximated using imputation. PCOP are points rather than regions. If we are to develop methods specific for more distant ancestors, we need at least two things: 1) good methods to distinguish small elements and 2) small elements that are stable across generations. We have some clues about the opportunity in the current work and in recent science. At present the opportunity is aspirational. But a roadmap is emerging.

Based on the experiences in the current work, a few suggestions for future efforts are offered. GFG can be augmented by further additions of UDFs and additional graphs. In personal work, the author uses graph methods to link coordinates in an image (photograph, historical document, etc.) to family tree members and others, which add value by 1) forming family/friends, associates, and neighbor (FAN) groups, 2) linking source documents as evidence, 3) adding temporal data, and 4) in many cases adding geographic graphs . There are extant genealogy timeline tools[48] which might be augmented by graph methods. Genealogists might emulate personal trajectories developed in healthcare[49]. Geographic data is tessellated into graphs which can be linked to genealogy events and thereby incorporated into analytics. Neo4j has a set of spatial functions[50],[51] that have more recently been expanded in a spatial plugin[52] to support geo-analytics and use of shape files such as the Newberry Library Atlas of Historical County Boundaries[53]. While GFG-PI support importing the FTDNA Y-haplotree and raw Y-DNA csv files, Y- and mt-DNA management and analytics are not well developed yet in GFG. Collaboration between research teams can be well managed in a graph ecosystem. GFG can be refined to recommend best opportunities for candidates to join projects or doing DNA testing. Forensic genealogy could incorporate law enforcement graphs to connect families, police, victims, perpetrators, and DNA enabling traversals to discover actionable insights.

Testing the monophyletic segment hypothesis is required. Current vendor data is not harmonized, producing aligned but slightly different segment boundaries from the same raw SNP data. Large datasets are required to identify MSS but the precise threshold numbers for success are not known. While the specificity of MSS was demonstrated using up to 250,000 DNA matches, this specificity could deteriorate in larger studies. More work is required to explore the biological basis for monophyletic segments.

One of the challenges for graph aficionados in the paucity of graph thinking in our genealogy community. There are, of course, notable exceptions. But native graph databases (not no-SQL) are not deployed widely by genealogy service providers. Unfortunately, hierarchical data standards (XML, JSON, OWL, etc.) aligned with interoperability of graph data were pushed aside in developing an updated GEDCOM 7 standard. Previous work on graphs in genealogy by citizen scientists was focused on visualizations which are interesting but difficult to translate into actionable recommendations. GFG is collaborating with GraphXR[28] on developing 3D renderings, including virtual and augmented reality outputs[54].

Acknowledgments

The author received no financial support for this project. The opinions expressed here are those of the author and may not reflect those of many consulted colleagues. The author is grateful to his co-administrators of several FTDNA surname projects, their participants, and expert genealogists who contributed to the data required for this project, specifically Charles Stinnett, Lezlie Russell Markey, Joe Stennet, John Stennett, Michelle

Pendleton, and Laree Lee (Stinnett Project); Terry Avitts, Jean Seeley, and Elizabeth Cartwright (Avitts Project); Melle van der Heide (Teves Project); and James Irvine, Peter Irvine, and James Noble (Clan Irvine Project). Marty Acks was a helpful alpha tester of GFGS and the GFG=PI. Valuable critiques were offered by Debbie Kennett, Evert Jan Blom, and Tim Janzen. I have benefitted from weekly Neo4j discussions through Northwestern University with Mengia Kahn, Narcisco Albarracin and Weidong Yang.

Conflicts of Interest

The author is the principal at Who Am I, LLC which distributes the Graphs for Genealogy software.

References

[1] "Neo4j Included in Gartner's Magic Quadrant for Data Management Solutions for Analytics 2018," *Neo4j Graph Data Platform*. https://neo4j.com/news/neo4j-included-in-gartners-magic-quadrant-for-data-management-solutions-for-analytics-2018/ (accessed Mar. 18, 2022).
[2] "Cypher Query Language - Developer Guides," *Neo4j Graph Database Platform*. https://neo4j.com/developer/cypher/ (accessed Jan. 02, 2022).
[3] "Graph Query Language GQL." https://www.gqlstandards.org/home (accessed Jan. 04, 2022).
[4] "Awesome Procedures On Cypher (APOC) - Neo4j Labs." https://neo4j.com/labs/apoc/ (accessed Jan. 02, 2022).
[5] "Neo4j Graph Data Science - Developer Guides," *Neo4j Graph Database Platform*. https://neo4j.com/developer/graph-data-science/ (accessed Jan. 02, 2022).
[6] "Neo4j in Action," *Manning Publications*. https://www.manning.com/books/neo4j-in-action (accessed Jan. 02, 2022).
[7] "The Next-Generation Graph Database Built for Unlimited Scale and Developer Flexibility," *Neo4j Graph Data Platform*. https://neo4j.com/product/neo4j-graph-database/scalability/ (accessed Mar. 18, 2022).
[8] "Neo4j Download Center," *Neo4j Graph Database Platform*. https://neo4j.com/download-center/ (accessed Jan. 02, 2022).
[9] D. A. Stumpf, "GFG Software," *Who Am I*. https://www.wai.md/product-page/gfg-software (accessed Jan. 02, 2022).
[10] D. A. Stumpf, "Neo4j Genealogy PlugIn jar file," *Neo4j Genealogy PlugIn jar file*, Dec. 31, 2021. https://github.com/waigitdas/Neo4j-Genealogy-PlugIns/tree/main/jar (accessed Jan. 02, 2022).
[11] D. A. Stumpf, "Installing Neo4j." https://help.gfg.md/gfg-software/installingNeo4j.html (accessed Feb. 06, 2022).
[12] "Setting up a plugin project - Java Reference," *Neo4j Graph Database Platform*. https://neo4j.com/docs/java-reference/4.4/extending-neo4j/project-setup/ (accessed Jan. 02, 2022).
[13] D. A. Stumpf, "Neo4j Genealogy PlugIn pom file," Dec. 31, 2021. https://github.com/waigitdas/Neo4j-Genealogy-PlugIns/blob/main/pom.xml (accessed Jan. 02, 2022).
[14] D. A. Stumpf, "Neo4j Genealogy PlugIn java class files," Dec. 31, 2021. https://github.com/waigitdas/Neo4j-Genealogy-PlugIns/tree/main/java%20classes (accessed Jan. 02, 2022).
[15] D. A. Stumpf, "Graph Databases in Genealogy," Sep. 30, 2017. http://stumpf.org/genealogy-blog/graph-databases-in-genealogy (accessed Jan. 03, 2022).
[16] *Genealogy & Family Trees - Discover Neo4j AuraDB Free with Michael and Alexander*, (Jan. 31, 2022). Accessed: Feb. 04, 2022. [Online Video]. Available: https://www.youtube.com/watch?v=EWdb16ibgG8
[17] "GRAPHical Family Trees," *Findmypast Tech*, Feb. 20, 2018. https://tech.findmypast.com/graphical-family-tree/ (accessed Feb. 04, 2022).

[18] "Full-text search index - Neo4j Cypher Manual," *Neo4j Graph Data Platform*. https://neo4j.com/docs/cypher-manual/4.4/indexes-for-full-text-search/ (accessed Apr. 29, 2022).
[19] B. T. Bettinger, "The Shared cM Project Version 4.0 (March 2020)," p. 56, 2020.
[20] D. A. Stumpf, "File Setup," *Preparing your data*. https://help.gfg.md/gfg-software/UserFilesSetup.html (accessed Feb. 06, 2022).
[21] "LOAD CSV - Neo4j Cypher Manual," *Neo4j Graph Database Platform*. https://neo4j.com/docs/cypher-manual/4.4/clauses/load-csv/ (accessed Jan. 02, 2022).
[22] "Load CSV - APOC Documentation," *Neo4j Graph Database Platform*. https://neo4j.com/labs/apoc/4.1/import/load-csv/ (accessed Jan. 02, 2022).
[23] "FTDNA Y-haplotree JSON file," Jan. 02, 2020. https://www.familytreedna.com/public/y-dna-haplotree/get (accessed Jan. 02, 2022).
[24] "HapMap HG37 tab-delinited files," *GitHub*. https://github.com/waigitdas/Neo4j-Genealogy-PlugIns (accessed Jan. 03, 2022).
[25] J. F. Curran, M. C. Crane, and J. H. Wray, *Numbering Your Genealogy: Basic Systems, Complex Families, and International Kin*. Arlington, Va.: Natl Genealogical Society, 2008.
[26] P. O'Neil, E. O'Neil, S. Pal, I. Cseri, G. Schaller, and N. Westbury, "ORDPATHs: insert-friendly XML node labels," in *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, New York, NY, USA, Jun. 2004, pp. 903–908. doi: 10.1145/1007568.1007686.
[27] D. N. A. Painter, "DNA Painter." https://dnapainter.com (accessed Mar. 02, 2022).
[28] "Visualization," *Kineviz*. https://www.kineviz.com/visualization (accessed Mar. 24, 2022).
[29] "Export to Gephi - APOC Documentation," *Neo4j Graph Data Platform*. https://neo4j.com/labs/apoc/4.1/export/gephi/ (accessed Apr. 29, 2022).
[30] "Hardware Sizing Calculator," *Neo4j Graph Data Platform*. https://neo4j.com/hardware-sizing/ (accessed Mar. 11, 2022).
[31] "Downloads | Notepad++." https://notepad-plus-plus.org/downloads/ (accessed Mar. 18, 2022).
[32] B. Bettinger, "A Small Segment Round-Up," *The Genetic Genealogist*, Dec. 29, 2017. https://thegeneticgenealogist.com/2017/12/29/a-small-segment-round-up/ (accessed Mar. 18, 2022).
[33] "AncestryDNA® ThruLines®." https://support.ancestry.com/s/article/AncestryDNA-ThruLines?language=en_US (accessed Feb. 12, 2022).
[34] A.-L. Barabási, *Network Science*, 1st edition. Cambridge, United Kingdom: Cambridge University Press, 2016.
[35] "Monophyletic Segments," *Who Am I*. https://www.wai.md/post/monophyletic-segments (accessed Sep. 19, 2022).
[36] B. Bettinger, "A Triangulation Intervention," *The Genetic Genealogist*, Jun. 20, 2016. https://thegeneticgenealogist.com/2016/06/19/a-triangulation-intervention/ (accessed May 07, 2022).
[37] D. Gosnell and M. Broecheler, *The Practitioner's Guide to Graph Data: Applying Graph Thinking and Graph Technologies to Solve Complex Problems*, 1st edition. Sebastopol, CA: O'Reilly Media, 2020.
[38] "The Neo4j Graph Data Science Library Manual v2.0 - Neo4j Graph Data Science," *Neo4j Graph Database Platform*. https://neo4j.com/docs/graph-data-science/current/ (accessed May 10, 2022).
[39] L. Neill, "Building a Graph of History with The Codex," *Neo4j Graph Data Platform*, Aug. 05, 2020. https://neo4j.com/blog/building-graph-history-codex (accessed Feb. 26, 2022).
[40] J. Bartlett, "Lessons learned from triangulating a genome.," in *Advanced Genetic Genealogy. Techniques and Case Studies.*, D. Parker-Wayne, Ed. Cushing, TX: Wayne Publishing, 2019, pp. 1–15.
[41] T. Misteli, "The Self-Organizing Genome: Principles of Genome Architecture and Function," *Cell*, vol. 183, no. 1, pp. 28–45, Oct. 2020, doi: 10.1016/j.cell.2020.09.014.
[42] A. D. Broido and A. Clauset, "Scale-free networks are rare," *Nat. Commun.*, vol. 10, no. 1, Art. no. 1, Mar. 2019, doi: 10.1038/s41467-019-08746-5.

[43] G. Lima-Mendez and J. van Helden, "The powerful law of the power law and other myths in network biology," *Mol. Biosyst.*, vol. 5, no. 12, pp. 1482–1493, Nov. 2009, doi: 10.1039/B908681A.
[44] *Do Chromosome Segment End Points Matter? | Genetic Genealogy Explained*, (Feb. 03, 2021). Accessed: Mar. 20, 2022. [Online Video]. Available: https://www.youtube.com/watch?v=3vWBeFUBVdg
[45] R. Hu *et al.*, "Family Finder Matching 5.0. Matching Algorithm and Relationship Estimation. White Paper 2021-08-18." FAmily Tree DNA, Aug. 18, 2021. [Online]. Available: https://blog.familytreedna.com/wp-content/uploads/2021/08/Family_Finder_Matching_WhitePaper.pdf
[46] jim4bartletts, "How Many TGs From Distant Ancestors?," *segment-ology*, Dec. 17, 2019. https://segmentology.org/2019/12/17/how-many-tgs-from-distant-ancestors/ (accessed Mar. 24, 2022).
[47] B. M. Henn *et al.*, "Cryptic Distant Relatives Are Common in Both Isolated and Cosmopolitan Genetic Samples," *PLoS ONE*, vol. 7, no. 4, p. e34267, Apr. 2012, doi: 10.1371/journal.pone.0034267.
[48] "Cyndi's List - Timelines - General Resources." https://www.cyndislist.com/timelines/general/ (accessed Mar. 24, 2022).
[49] "Improving Patient Outcomes with Graph Algorithms," *Neo4j Graph Data Platform*, Jun. 24, 2020. https://neo4j.com/blog/improving-patient-outcomes-algorithms-graphconnect (accessed Mar. 24, 2022).
[50] "Spatial functions - Neo4j Cypher Manual," *Neo4j Graph Data Platform*. https://neo4j.com/docs/cypher-manual/4.4/functions/spatial/ (accessed Mar. 24, 2022).
[51] "Spatial - APOC Documentation," *Neo4j Graph Data Platform*. https://neo4j.com/labs/apoc/4.1/misc/spatial/ (accessed Mar. 24, 2022).
[52] "Neo4j Spatial." Neo4j Contrib, Mar. 07, 2022. Accessed: Mar. 24, 2022. [Online]. Available: https://github.com/neo4j-contrib/spatial
[53] "Home | Atlas of Historical County Boundaries Project." https://digital.newberry.org/ahcb/index.html (accessed Mar. 24, 2022).
[54] "Installing WebXR Virtual Reality for GraphXR." https://helpcenter.kineviz.com/user-guides/HC/Installing-WebXR-Virtual-Reality-for-GraphXR.1150615943.html (accessed Mar. 21, 2022).

Appendices

Appendix 1	GFG PlugIn jar file	 wai.neo4j.gen-1.0.3.jar	10
Appendix 2	Unzip in c:/genealogy/neo4j/	 wai_files.zip	20
Appendix 3	List of UDF in GFG-PI	See Supplement 1. Tab Functions	22
Appendix 4	GEDCOM-Kit curation file template	 GEDCOM-DNA_curation.csv	30
Appendix 5	Triangulation group template	 TG_Template.csv	40
Appendix 6	Data Dictionary (Neo4j Schema)	 data_dict_20220508_112934.csv	60
Appendix 7	Database indices	 Neo4j indices.csv	70
Appendix 8	Grass files: unzip and place in a convenient folder	 gfg_grass.zip	80
Appendix 9	Database properties	 properties.xlsx	90

Supplemental Data

Supplement 1	Know your data	 Know_your_data_20 220311_070316-ano	10
Supplement 2	X-chr descendancy tree	 x-chr descendancy tree.xlsx	20
Supplement 3	DNA Painter MSS segments	 DNA Painter Stinnett MSS anony	39
Supplement 4	Y-haplotree	 Y-haplotree.svg	50

[1] "HapMap HG37 Tab-Delimited Files," GitHub, accessed January 3, 2022, https://github.com/waigitdas/Neo4j-Genealogy-PlugIns.