Graph-Based Remerging of Genealogical Databases

D. Randall Wilson
fonix Corporation
WilsonR@fonix.com

1. Introduction

Computers have helped family history research tremendously, and personal genealogy programs help people organize their information in a way not previously possible. However, most personal genealogy software packages available today suffer from weak merging functionality, which makes collaboration with others much more difficult than it needs to be. In particular, if someone shares a copy of their electronic genealogical database with a relative, and then both make additions and/or changes to the data, there is currently no reasonable way in most genealogical programs to merge these changes together. This paper reviews current approaches to this problem and presents an algorithm for taking much of the agony out of the merging process by taking into account the relationships of individuals in the database.

2. Current Approaches

Several approaches been used to address the problem of merging independent changes to a genealogy database back together. This section lists a few of the more common methods.

None. One of the most common solutions is to give up. Often one person in the family becomes the only one who maintains a genealogy database, and everyone gives information to them. This places a large burden on one person while leaving the others largely uninvolved. Alternatively, several people in the family each build their own database and repeat all of the work done by others. In either case, efforts are duplicated rather than distributed.

Visual Inspection. Another approach is to visually inspect the two databases (or large printouts from them) to find differences that are then entered by hand. This is, of course, very tedious, time-consuming and error-prone. One person reported that they used the Unix *diff* command to compare GEDCOM files exported from the two databases in order to find any differences, and then typed the additions in by hand.

Match/Merge. Perhaps the most common method for doing merging is to use a programs "Match/Merge" function, which typically imports all of the second database's information into the first, and then forces the user to look at every pair of similar individuals and decide if they are really the same. When the two databases originated from a common database, the vast majority of individuals will in fact match, but there will be some differences, thus requiring careful and tedious examination of almost every name in the entire database, even if only a few names have changed. Worse, the process is error-prone, and mistakes can either merge individuals together that should not be (e.g., children with the same name because one died young), or leave multiple copies of an individual dangling in the database. This process is so cumbersome that it often requires much more work to incorporate someone else's changes into a database than it does to simply re-enter the data by hand, often defeating the purpose of having someone else help with data entry and/or research.

Some programs at least keep the "imported" data separate from the "original" data so that individuals in the original database are not considered for matching with each other, but this does not solve the problem of having to examine everyone in the database.

Checking Out. At least one program allows one database to be designated as the "master" copy, and provides for other people to "check out" portions of the family tree that only they are allowed to modify. While this does allow independent updates to the database, it constrains what parts of the family tree each person can work on. It also requires careful coordination of efforts and thus is more restrictive than necessary.

Unique ID Numbers. A few genealogy programs are starting to use a better solution to this problem, which is to use a unique ID number for each individual to automatically identify matches upon remerging. Using ID numbers greatly helps in this problem, but still has a few weaknesses. First, it does not help those who have *already* shared their data with someone else (i.e., before the ID number solution was available). Secondly, the ID numbers may not survive translations if the data is, for example, exported to a GEDCOM file and read into another genealogy program and then brought back in as another (updated) GEDCOM file, as might be the case if two people working on the same database use different software and/or different computer platforms.

3. Graph-based Merging Algorithm

A new algorithm is presented here for aiding in the merging process by taking into account the relationships of individuals in the database. The algorithm does not require the storage of unique ID numbers (though ID numbers can add an extra level of confidence if they are available), nor does it require anyone to "check out" (lock) portions of the database. Users are free to make independent updates and then upon receiving a copy of another version of the database, they can view the differences fairly quickly and decide which pieces of information, individuals, or new branches of individuals they wish to import into their database, and resolve any conflicts that have arisen.

Outline of Algorithm. Suppose two people begin with a copy of the same database. Both modify their databases, resulting in two databases that have much information in common. Then one person wants to merge the second one (the *incoming* database) into the first one (the user's *own* database) in order to create a "complete" database with all of the information from both of them.

The algorithm begins by identifying groups of individuals that appear in both databases that can be assumed (with very high probability) to be matches, and then presents any additions or conflicts to the user so they can incorporate the new information into their database. The algorithm proceeds as follows.

1. Sort both databases using the same criteria (name, birth date, etc.—even unique ID numbers if available).
2. Find matching individuals.
   a) Start at the top of both sorted lists.
   b) As long as either person is "less than" the other, skip the "lower" one. Also, skip anyone that is already part of a "subgraph."
   c) When a match is found:
      i. Mark both people as part of the same "subgraph".
      ii. Recursively find relatives of these people (parents, spouses, children) that also match, marking each of them as part of the same subgraph.
      iii. Keep a list of differences encountered at the "edges" of the subgraph to come back to.
      iv. Continue until all relationship links have ended in differences.
   d) Continue Steps (b) and (c) until the bottom of both lists are reached.
3. Choose the largest subgraph (i.e., the one with the most individuals in it) and make sure it is large enough to ensure that the matches are true (e.g., at least 20 people or 20% of the database).
4. Present the user with the differences at the edges of this subgraph.
   a) For *additional individuals* (i.e., those in the incoming database but not in the user's own database), the user can decide whether to add them (and/or whether they are the same person as someone else in the same family). The user may also choose to add any other additional individuals that connect through this one in order to add whole subgraphs at once.
   b) For individuals with *additional information* (e.g., a death date, ordinance information, or additional notes), the user can decide whether to include the information.
   c) For individuals with *conflicting information* (e.g., a different birth place), the user can decide which information to use in their version of the individual, and whether to

mention the difference in a note. The user could also decide that the individuals with conflicting information are in fact different people, in which case the extra person can be added or excluded.

   d) When the incoming database has *missing individuals* or otherwise matching individuals that are missing fields of information, the user need not be bothered, since the idea is to add to their database rather than delete from it. However, the user may choose to be shown individuals with missing information in order to confirm that the match is valid. Furthermore, the user may choose to be shown missing individuals with an option to delete them, if they suspect that the incoming database may have removed some individuals that should really not have been in there.

5. Once an individual is added or modified to bring the two databases into agreement and have the user's confirmation that they reference the same person, the individuals are considered "matches" (even if they still have some differences). If the individual was on the boundary between two subgraphs, then the two subgraphs may be combined (i.e., people in the other one are marked as being part of this one, and the list of "edge differences" are added together). In this way, the database can often eventually become one large graph that includes everyone. Steps 3-5 are repeated on the largest remaining subgraph until all individuals in the incoming database are accounted for.

This algorithm avoids adding anything to the main database without the approval of the user, but also saves the user from having to deal with the vast majority of the records which are clearly identical. It is common to have several people in the same database with the same name, and possible that their other information will match, too (especially if they do not have much information specified). However, it is extremely unlikely that they will both have the same relationship to dozens of people whose information also matches unless they are really the same person. The key to this algorithm is the use of relationships between similar people to increase the confidence of matches.

Many users prefer to examine any new information being added to their precious database, but others may trust the incoming database enough to prefer more automation. They may prefer to have all new people and/or all new fields of information added to their database without having to examine each one, and then deal personally only with the few cases where there are conflicting pieces of information (e.g., different birth dates for an otherwise identical individual).

4. Uses of Graph-Based Merging

This algorithm is useful in a variety of situations. Once someone shares their database with family members, it allows everyone to make independent updates as they each do research. It allows family members to add information on their immediate family and then more easily share this information with others. Similarly, it allows an archivist of a family history organization to assign work to others, to make updates concurrently with the research director and others in the organization, and receive updates to portions of the family that certain people in the organization know more about (such as their own close relatives).

The algorithm also allows people to have copies of their database on multiple computers (e.g., their main desktop machine as well as their laptop computer), make changes in either one, and then quickly merge the changes back together. It can be used "retroactively" on databases that have already diverged from a common source before this feature was even made available. It could also be used to report a concise list of differences between two databases.

It can be used to combine two partially-overlapping databases, as might exist if someone created a seperate database for each of their two parents and wanted to link them together while merging all information in both databases regarding the immediate family members common to both databases.

The algorithm could also be used to add information into a database that was previously ommitted. For example, if information is exported from one genealogy program to another, but certain fields or individuals were ommitted, and then changes are made to the new version of the database, the ommitted fields could later be merged into the second database.

While there are other uses beyond these, the above uses are all ones that the author has personally come across needs for just in the last year.

5. Conclusion

This algorithm greatly reduces the burden placed upon users when remerging genealogical databases. It is straightforward enough to be added to almost any genealogy program without modifying the stored data structure, and could also be used in standalone utilities. It does not require ID numbers nor the “checking out” of portions of the tree, and can thus be used retroactively on existing genealogical databases that have already diverged from a common source. It allows the user to retain control over what information is added to their database while requiring an amount of effort that is proportional to the amount of information being imported rather than the size of the two database in question.

Existing merging algorithms are still useful in some situations (such as finding duplicate individuals that exist in the same database), so the proposed algorithm should add to rather than replace such tools.

The use of this algorithm would make it practical for individuals and family organizations to have multiple people working together on the same genealogical database and share their updates much more elegantly that is currently allowed by most software.