Software for Constructing and Verifying Pedigrees Within Large Genealogies and an Application to the Old Order Amish of Lancaster County

Richa Agarwala[*] LGDR/NHGRI *Leslie G. Biesecker*[†] LGDR/NHGRI *Katherine A. Hopkins*[‡] JHU

Clair A. Francomano[§] MGB/NHGRI *Alejandro A. Schäffer*[¶] LGDR/NHGRI

January 30, 1998

Keywords: Genealogy, pedigree, linkage analysis, Amish, database, algorithm, Steiner tree

Address for correspondence: Alejandro A. Schäffer; National Human Genome Research Institute; National Institutes of Health; Suite 2000; 333 Cassell; Baltimore, MD 21224;
FAX: (410) 550-7513

[*]National Institutes of Health, Bethesda, (under contract to R.O.W. Sciences, Inc., Rockville), `email:richa@helix.nih.gov`

[†]National Institutes of Health, Bethesda, `email:leslieb@helix.nih.gov`

[‡]Immunogenetics Laboratory, Division of Medical Genetics, Department of Medicine, Johns Hopkins University, School of Medicine, Baltimore, `email:Kathy.Hopkins@path-online.path.jhu.edu`

[§]National Institutes of Health, Bethesda, and Center for Medical Genetics, and Department of Medicine, Johns Hopkins University, School of Medicine, Baltimore, `email:clairf@nhgri.nih.gov`

[¶]National Institutes of Health, Bethesda and Baltimore, `email:schaffer@helix.nih.gov`

Abstract

This paper describes PedHunter, a software package that facilitates creation and verification of pedigrees within large genealogies. A frequent problem in medical genetics is to connect distant relatives with a pedigree. PedHunter uses methods from graph theory to solve two versions of the pedigree connection problem for genealogies as well as other pedigree analysis problems. The pedigrees are produced by PedHunter as files in LINKAGE format ready for linkage analysis. PedHunter uses a relational database of genealogy data, with tables in specified format, for all calculations. The functionality and utility of PedHunter are illustrated by examples using the Amish Genealogy Database (AGDB), a database created for the Old Order Amish community of Lancaster County.

Introduction

Many genetic studies require linkage analysis, a key step towards positional cloning of disease susceptibility genes. Linkage analysis requires describing pedigrees for a set of people exhibiting a specific trait and verifying relationships in pedigrees (Polymeropoulos et al. 1996; Khoury et al. 1987). When a pedigree is implicitly part of a large genealogy, it is possible to use computer science techniques to find a pedigree. There may be many potential (sub)pedigrees that connect the individuals with the trait, and it is desirable to find a (sub)pedigree that is optimal in a rigorous mathematical sense. This paper describes software tools, called **PedHunter**, to keep genealogical data in a relational database and to analyze the data systematically. PedHunter solves two distinct formulations of the optimal pedigree connection problem, as well as other problems related to pedigree construction, verification, and analysis.

Closed populations are a rich source of material for medical genetics studies. For example, closed populations provide powerful data sets to map recessive disease genes by homozygosity mapping (Kruglyak et al. 1995). Some closed populations have compiled genealogies due to their interests, and these genealogies enhance the utility of the populations for genetic studies. The Old Order Amish of Lancaster County in Pennsylvania are one such population. Most ancestors of the Amish in the United States migrated in early eighteenth century from Germany and Switzerland. They settled in Berks County, Pennsylvania and gradually radiated to other parts of Pennsylvania, and Ohio, Indiana, Illinois, and Iowa. Some settled in Lancaster County, Pennsylvania in about 1795. Information about Amish pedigrees of Lancaster County in Pennsylvania is available in a book called the "Fisher Family History"(Fisher Family History, 1988), also known as the "Fisher book", which traces the lineage of the Amish in magnificent detail. In spite of the rich information in this book, it is not easy to use in medical genetics for several reasons.

The limitations of the Fisher book include:

1. The (1988 version) book has about 550 pages of information in small font.
2. The organizers grouped information about extended families in the book. This ordering is useful for Amish readers interested in their proximal genealogy, but is problematic for the construction of extended pedigrees needed for genetic analysis.
3. Many people have the same first and last name.
4. There are no pedigree diagrams, only text.
5. Some information about a person may appear in several places because that person has his own record and also appears in the records of parents and children. Data is sometimes inconsistent across multiple entries.

Due to these limitations, tracing family relationships for genetic studies requires laborious searching. This paper describes a database called **Amish Genealogy Database (AGDB)** for the genealogy of the Old Order Amish of Lancaster County, Pennsylvania. AGDB contains information from the 1988 version of Fisher Family History that has been corrected for errors and contains additional information compiled subsequent to

1988. Keeping the data in AGDB and querying the data with PedHunter facilitates usage of the genealogy by medical geneticists.

Beyond the Old Order Amish, PedHunter is a general tool that can be used to analyze any genealogy. The use of PedHunter requires only a database with information for that population. For confidentiality reasons, no phenotype information is stored in the database. Phenotype information known only by the user may be helpful in posing genetically interesting queries.

Tools from Computer Science

This section summarizes relevant data storage and data analysis tools from the domain of computer science and suggests how they can be used to make a genealogy database. The data in AGDB are stored in a *relational database* (Date 1990). PedHunter is designed to use relational databases developed in SYBASE. SYBASE provides an implementation of Structured Query Language (SQL) useful for answering simple queries and an interface with the C programming language useful for implementing more complex queries.

Data organization

Each record in the Fisher book contains information about a nuclear family. For example,[1]

894. Noah Karp b Jan. 8, 1938, d May 9, 1988, farmer, m Dec. 24, 1957, Mary Green (4211) b Sep. 12, 1940, d 1962. Children: Victor (895), Sarah (4200), Amy. m2 1964, Annie Green (4211) b Aug. 23, 1942. Children: Henry.

895. Victor Karp (894) b May 24, 1959, m June 6, 1982, Mildred Sweeney.

4211. John Green b Dec. 24, 1920, m Sarah Fisher. Children: Annie (894), Mary (894).

The data were obtained as an ASCII text file with some typesetting commands inserted. Using standard methods from the theory of formal languages these family records were parsed and collated into a record of the information for each person.

Each record in the Fisher book contains the following pieces of information relevant to genealogies:

- Name of family head; e.g., *Noah Karp, Victor Karp*, and *John Green* in the above three examples, respectively.
- A number called the Fisher number or family number. For the above examples, the Fisher numbers are *894, 895*, and *4211*, respectively.
- If the parents of a family head are known, the record contains the Fisher number for that family; e.g., record *895* refers to record *894*, Victor's parents.

[1]Information in this example of a family record is fictitious, but the syntax is accurate.

- List of spouses. For each spouse, the record gives the name and attempts to give a Fisher number in following order: (i) Fisher number for the family in which spouse is the family head (ii) Fisher number for the family in which spouse is a child. For example, in record *894* Noah is married first to Mary and then to Annie Green; the Fisher number *4211* following them refers to the family in which they were children.

- List of children of each spouse. This list contains name of children and attempts to give a Fisher number for each child in following order: (i) Fisher number for the family in which child is family head e.g., *895* for *Victor Karp* in record *894* (ii) Fisher number for the family in which child is a spouse; e.g, *894* for both Annie and Mary in record *4211*.

Data inconsistencies

The analysis of data with a relational database requires: (1) resolving or deleting inconsistencies in the data, (2) designing the database, (3) creating the tables, and (4) writing programs to analyze the database. Of these four steps, the only task that is specific to a population is deleting or resolving inconsistencies. The other three tasks are greatly facilitated by using a commercial SQL implementation. Finding inconsistencies is feasible if the raw genealogy data has redundant and conflicting information.

Due to the structure of the records in the Fisher book, the detectable inconsistencies include parent-child relationships and husband-wife relationships. Some inconsistencies are caused by information that is present in one record, but absent in another; it is assumed that the information is correct in this case. Other inconsistencies are apparently caused by typographical errors in Fisher numbers, in which case it may be straightforward to decide which numbers are (in)correct. A disproportionate number of the errors occur in the data entered after 1988, presumably because those data have not been published and hence have not been proofread by large numbers of Amish. In addition, the number of recorded births, deaths, and marriages per year dropped precipitously after 1988, suggesting that much recent information is missing. Efforts have been initiated with the Amish community to correct remaining errors and update records.

Database design

Each person in the database is assigned a unique integer id, called the **Program id**. There are two central tables in the database – the person table and the relationship table. The person table has information exclusive to the individuals and the relationship table keeps information relating the individuals by containing an entry for each couple and a list of their children, if any, in terms of their program id. Please see Appendix for details of database design.

Analyzing the database

After the database is created, the next step is to use analytic tools to extract useful information. The analytic tools in PedHunter rely heavily on graph theory(Even 1979). In the Appendix, the notion of a graph is introduced along with related terminology.

Results and Examples

This section describes the queries available in PedHunter and shows the utility of PedHunter with a usage of AGDB on a published pedigree. PedHunter supports functions for finding relatives of individuals (father, mother, siblings, half siblings, uncles and aunts, children, first cousins, ancestors) as well as functions for testing if two individuals are related according to a specified relationship. The more complicated queries supported by PedHunter are as follows:

- *lca(S):* [lowest common ancestors for S] This query finds most recent common ancestors of the given set of people S; the set of most recent common ancestors of S is the set of people A such that (i) everyone in A is an ancestor of everyone in S and (ii) no one in A has a descendant who is also an ancestor of everyone in S.

- *asp(S):* [all shortest paths pedigree for S] If there is at least one common ancestor for the given set of people S, then for each lowest common ancestor a, *asp(S)* finds the pedigree containing all shortest paths between a and each person in S.

  PedHunter contains an implementation of Dijkstra's Algorithm (Dijkstra 1959) for finding all shortest paths. PedHunter takes a few seconds for creating modest size pedigrees (e.g., connecting 85 to 90 individuals resulting in a pedigree of about 500 individuals). This is a significant improvement over the tens of hours typically required to create such pedigrees by manually searching the Fisher book.

- *minped(S):* [minimal pedigree for S] If there is at least one common ancestor for the given set of people S, then for each lowest common ancestor A, *minped(S)* finds a pedigree connecting A and each person in S such that (i) the pedigree has minimum number of vertices and (ii) every path from A to each person in S is a shortest path.

  The problem of finding a minimal pedigree for S is the Steiner tree problem with *asp(S)* being the given graph and S being the set of vertices to be connected. PedHunter uses a "branch and bound" algorithm for finding a minimal pedigree. This is described in the Appendix.

- *pedigree_length(X, N):* Prints a pedigree containing all descendants of X who are at most N generations apart from X. For example, *pedigree_length(X, 2)* lists all children and grandchildren of X.

- *kinship(X, Y):* Computes the kinship coefficient between X and Y where *kinship(X, Y)* is the probability that a randomly selected allele of X and a randomly selected allele of Y are identical by descent. PedHunter uses the algorithm in (Weir 1996, pp. 204–206).

- *inbreeding(X):* Computes the inbreeding coefficient for X where *inbreeding(X)* is the probability that the two alleles of X are identical by descent. Since X inherits a randomly selected allele from mother and a randomly selected allele from father,

  $$\textit{inbreeding}(X) = \textit{kinship}(X\textit{'s mother}, X\textit{'s father})$$

  PedHunter uses the algorithm in (Weir 1996, pp. 204–206).

- *print(X):* Print all information about person X.

All pedigrees are produced in LINKAGE format (Terwilliger and Ott 1994). The PedHunter output pedigree files can be read into some versions of pedigree/genotype maintenance programs such as CYRILLIC 2.0 (Chapman 1990) and Pedigree/Draw 5.0 (Mamelka et al. 1993). Two of the drawings here were produced by feeding PedHunter output into Pedigree/Draw 5.0.

The all shortest paths pedigree may contain many connections between pairs of individuals, and it can be quite large. This pedigree may be too complicated for linkage analysis with current tools and may be impossible to draw comprehensibly (e.g., the drawing in (McKusick et al. 1964)). These two problems justify the desire for a minimal pedigree that is a sub-pedigree of the all shortest paths pedigree. A minimal pedigree can be used to explain most likely paths of inheritance of the mutant trait gene. A minimal pedigree can be used as a starting point for the geneticist to reinstate more individuals that are in the all shortest paths pedigree in an iterative linkage analysis.

To verify a given pedigree P, one must verify all paths in P. A path is considered to be *verified* if the parent-child relationships represented by the edges on the path are consistent with the information in the database. The paths in P can be verified as follows:

1. Find the set of people S in P which do not have a child in P and have at least one parent in P.

2. Find *asp(S)*. Paths in P that are present in *asp(S)* are verified. If all the paths in P are verified, the verification is complete.

3. For each "long" path p from u to v that is not yet verified, find *asp(R)* where the elements of R are u and v. If p is a path in *asp(R)*, p is verified. If all the paths in P are verified, verification is complete.

4. For each edge $u \rightarrow v$ in P that is not yet verified, do the query *is_child(v, u)*, which returns TRUE if v is a child of u and FALSE otherwise. If it returns FALSE, the edge is in error. (It can be corrected by using query features *father, mother, ancestor, children*, as appropriate for the pedigree. For instance, if confidence about v is high, u can be corrected by using *father(v)* and *mother(v)*.)

It is not practical to have a query feature for verifying an entire pedigree at once because it is difficult to determine what to do when there are multiple related errors in the pedigree. In many cases, the corrections would depend on knowledge not present in the pedigree database.

The utility of AGDB and PedHunter are demonstrated in an example concerning a genetic disorder currently under study. McKusick (1978) published a pedigree that connects children of 6 couples. Field work and consultation with the authors of the original study yielded the names of the persons, so they could be accessed in AGDB. Three of the couples are present in the Fisher book and AGDB and will be used as a test example. Part of the published pedigree that connects the 3 couples that are in AGDB is shown in Figure 2. The pedigree in Figure 2 is designated MKped. Let S be the set with elements $10, 16, 22, 26, 27, 32$ representing the 3 couples that are in AGDB. Using *lca(S)*, it turns out that there are 5 least common ancestors for the 3 couples. Figure 3 shows the all shortest paths pedigree (ASPped) for the least common ancestor that has maximum number of vertices in common with MKped. A minimal pedigree with the same least common ancestor is shown in Figure 4.

MKped (Figure 2) can also be used to illustrate pedigree verification. The paths in MKped that are verified by ASPped (Figure 3) are as follows:

- Path from 6 to 10.
- Path from 14 to 16.
- Path from 19 to 22.
- Path from 23 to 26 and 27.
- Path from 29 to 32.

According to ASPped 19 and 23 are siblings, whereas according to MKped 18 and 23 are siblings and 19 is a child of 18. This discrepancy is due to an error in MKped — person 18 should not be present in MKped.

The following relationships are verified using *is_child/siblings* query features:

- 5 is father of 6.
- 13 is father of 14.
- 17 is mother of 19 and 23.
- 28 is mother of 29.
- 17 and 28 are siblings.

Discussion

Much attention has been paid towards database management and analysis (e.g. GDB (Fasman et al. 1997), GenBank (Benson et al. 1997), SWISS-PROT (Bairoch and Apweiler 1997)) of genomes as well as to making extensive family trees (e.g. GEDCOM (1997), GED2HTML (1997)) but there is no tool sufficient for a medical geneticist who has a large genealogy as well as genetic information for some of the individuals. Existing software packages (e.g. CYRILLIC (Chapman 1990), DGENE (Lange et al. 1988), Pedigree/Draw (Mamelka at al. 1993), PEDRAW/WPEDRAW (Curtis 1990), PhenoDB (Cheung et al. 1996), SCHESIS (Round et al. 1989)) manage genetic information, store pedigrees, and draw pedigrees, but do not automatically extract or verify pedigrees from a genealogy. PedHunter provides tools to organize genealogical information in a database and a query facility to access the database. This functionality can create and verify pedigrees according to criteria specified by the user. Advantages of keeping data in a relational database include:

- *Simplicity:* The concept of a relation provides a self-contained means of representing all types of data relationships using simple tables.

- *Flexibility and restructuring:* There is complete flexibility in database design. Relations can be combined or broken down in other relations with ease.

- *Accessibility and Protection:* Keeping data in a relational database provides variable and ready access to all data. Data can be protected and can be used by multiple users.

- *Update:* Updating the information in a database is simple because one can verify whether information is consistent. It is important to keep data current for future use.

- *Optimization:* A good database design does not duplicate information. This reduces data storage and redundancy.

- *Easy Storage :* Physical data storage and logical database structure are independent. There is no worry about where and how the files are stored and they can be accessed using the relational tables.

The usage of PedHunter should be extensible to other existing genealogies besides the Fisher book. First, there are other genealogy books of the Old Order Amish that overlap with the Fisher book. Second, there are genealogy books of other Amish communities from the midwestern United States. Third, there are genealogy books of other closed North American communities, such as Hutterites and Mennonites. Fourth, there are genealogies loosely compiled for several island populations and for parts of Finland. Efforts have been initiated to obtain some of these genealogies, to obtain permission to computerize them, and to get the raw data into a computer-readable text file. To use PedHunter on a genealogy, one need only create a database with two tables – person_table and child_parent_table as described earlier and use the utility program provided to create generation_table.

PedHunter is freely available to scientists studying populations with written genealogies. Interested scientists should send an electronic mail request to **richa@helix.nih.gov** or to **schaffer@helix.nih.gov**. AGDB is intended to be available to scientists conducting IRB-approved research on the Old Order Amish. Access to AGDB is carefully controlled and limited by IRB protocol. In the initial phase, the IRB approved usage by 5 groups that have protocols approved by their IRBs to study the Amish. Each user group is represented by at least one co-investigator on the protocol for AGDB. More user groups meeting these criteria can be added.

The hunt for disease susceptibility genes has become increasingly automated. One step that has been lacking automated tools is that of connecting distant relatives with the same phenotype into a pedigree for linkage analysis. When the desired pedigree is implicitly part of a large, existing genealogy, computer science techniques can be adapted to extract the pedigree or to verify a putative pedigree systematically. PedHunter provides this automation. The utility of PedHunter is demonstrated by using it to query a database, AGDB, of a large Old Order Amish genealogy. These tools were used to verify and correct parts of a previously published pedigree. The combination of AGDB and PedHunter are immediately useful in ongoing genetic studies of this population.

Conversations with geneticists who use the Fisher book suggest that they are content when they can find *any* pedigree to connect a specified set of individuals. There is no reason to believe that the first pedigree found is the best. In contrast, the all shortest paths and Steiner criteria that PedHunter offers are mathematically rigorous and genetically sensible.

Materials and Methods

AGDB was created in Sybase SQL Server Release 11.0.x of the Sybase database management software. The functions are implemented using Transact-SQL and C version of Open Client DB-Library. PedHunter was developed on Unix System V Release 4.0 running under SunOS 5.5.1 using Sun WorkShop Compiler C 4.2, but it is compatible with other Unix computers. All the PedHunter functions are available as executable files that can be used from the command line prompt; AGDB is available to scientists conducting IRB-approved research on the Old Order Amish. The automatic compilation of the Fisher book genealogy into AGDB was approved by the National Cancer Institute Internal Review Board (IRB) at NIH.

Acknowledgements

This research was performed within the Division of Intramural Research, NHGRI. Thanks to Dr. Robert Nussbaum for frequent and active encouragement, for helpful guidance about the conventions of genetics research, and for suggestions on this manuscript. Thanks to Katie Beiler and our other Amish colleagues and friends in Lancaster County. Thanks to Gunther Birznieks and Ed Whitley for assistance with SYBASE. Thanks to Dr. Cyril Chapman, Kevin Crawford, and Dr. Paul Mamelka for their help with CYRILLIC and Pedigree/Draw. Thanks to Carol Machan for providing the raw data files for AGDB. The research of K.A.H. was supported by a grant NIH P01HG00373 from the National Center for Human Genome Research.

References

[1] Bairoch A. and R. Apweiler. 1997. The SWISS-PROT protein sequence data bank and its supplement TrEMBL, *Nucleic Acids Research*, **25** : 31–36.

[2] Benson, D. A., M. S. Boguski, D. J. Lipman, and J. Ostell. 1997. GenBank, *Nucleic Acids Research*, **25** : 1–6.

[3] Chapman, C. J. 1990. A visual interface to computer programs for linkage analysis, *American Journal of Medical Genetics*, **36** : 155–160.

[4] Cheung, K.-H., P. Nadkarni, S. Silverstein, J. R. Kidd, A. J. Pakstis, P. Miller, and K. K. Kidd. 1996. PhenoDB: An integrated client/server database for linkage and population genetics, *Computers and Biomedical Research*, **29** : 327–337.

[5] Curtis, D. 1990. A program to draw pedigrees using LINKAGE or LINKSYS data files, *Annals of Human Genetics*, **54** : 365–367.

[6] Date, C. J. 1990. *An Introduction to Database Systems (Vol. 1)*, New York, Addison-Wesley.

[7] Dijkstra, E. W. 1959. A note on two problems in connexion with graphs, *Numerische Mathematik*, **1** : 269–271.

[8] Even, S. 1979. *Graph Algorithms*, Computer Science Press, Rockville, Maryland.

[9] Fasman, K. H., S. I. Letovsky, R. W. Cottingham, and D. T. Kingsbury. 1997. The GDB(TM) Human Genome Database, *Nucleic Acids Research*, **25** : 72–80.

[10] Fisher Family History. 1988.

- *First edition (1957) compiled by* John M. Fisher family, *Published by* Amos L. Fisher.
- *Second edition (1972) edited by* Janice A. Egeland, *Published by* Moore Clinic, Johns Hopkins Hospital.
- *Third edition (1988)* revised and updated by a committee of Fisher descendants, Edited by Katie Beiler, *Published by* Eby's Quality Printing.

[11] GEDCOM. 1997. GEDCOM Coordinator - 3T, Family History Department, 50 East North Temple, Salt Lake City, UT 84150, USA. e-mail: gedcom@gedcom.org.

[12] GED2HTML: A GEDCOM to HTML Translator. 1997. Eugene W. Stark, 14 Landing Lane, Port Jefferson, NY 11777 USA. e-mail: stark@cs.sunysb.edu. URL: http://www.gendex.com/ged2html.

[13] Hwang, F. K. and D. S. Richards. 1992. Steiner tree problems, *Networks*, **22** : 55–89.

[14] Khoury, M. J., B. H. Cohen, E. L. Diamond, G. A. Chase, and V. A. McKusick. 1987. Inbreeding and prereproductive mortality in the Old Order Amish. III. Direct and indirect effects of inbreeding, *American Journal of Epidemiology*, **125** : 473–483.

[15] Kruglyak, L., M. J. Daly, and E. S. Lander. 1995. Rapid multipoint linkage analysis of recessive traits in nuclear families, including homozygosity mapping, *American Journal of Human Genetics*, **56** : 519–527.

[16] Lange, K., and R. C. Elston. 1975. Extensions to pedigree analysis. I. Likelihood calculations for simple and complex pedigrees, *Human Heredity*, **25** : 95–105.

[17] Lange, K., D. Weeks, and M. Boehnke. 1988. Programs for pedigree analysis: MENDEL, FISHER, and dGENE, *Genetic Epidemiology* **5** : 471–472.

[18] Mamelka, P. M., B. Dyke, and J. W. MacCluer. 1993. Pedigree/Draw for the Apple Macintosh, Population Genetics Laboratory Technical Report No. 1, Southwest Foundation for Biomedical Research, San Antonio, TX 78284, (1988, Second edition 1993).

[19] McKusick, V. A. 1978. The William Allan Memorial Award Lecture: Genetic nosology: Three Approaches, *American Journal of Human Genetics*, **30** : 105–122.

[20] McKusick, V. A., J. A. Egeland, R. Etheridge, and D. E. Krusen. 1964. Dwarfism in the Amish. I. The Ellis-van Creveld syndrome, Bull. Johns Hopkins Hospital, 115 : 306–336.

[21] Papadimitriou, C. H., K. Steiglitz 1982. *Combinatorial Optimization: Algorithms and Complexity* Prentice-Hall, Englewood Cliffs, NJ.

[22] Polymeropoulos, M. H., S. E. Ide, M. Wright, J. Goodship, J. Weissenbach, R. E. Pyeritz, E. O. Da Silva, R. I. Ortiz De Luna, and C. A. Francomano. 1996. The gene for the Ellis-van Creveld syndrome is located on chromosome 4p16, *Genomics*, **35** : 1–5.

[23] Round, A. P. 1989. SCHESIS: computation in a multipoint linkage analysis program, Progress in Clinical and Biological Research, 329 : 69–74.

[24] Terwilliger, J. D. and J. Ott. 1994. *Handbook of Human Genetic Linkage*, The Johns Hopkins University Press, Baltimore.

[25] Weir, B. S. 1996. *Genetic Data Analysis II*, Sinauer Associates, Sunderland, MA.

[26] Zelikovsky, A. 1997. A series of approximation algorithms for the acyclic directed Steiner tree problem, Algorithmica, 18 : 99–110.

Appendix

This Appendix describes the database design, graphs and related terminology, and method used by PedHunter to solve the *minimal pedigree*, or *Steiner pedigree* problem.

Database design

Each person in AGDB is assigned a unique integer id, called the **Program id**. Utility programs were written to translate from a Fisher id to the program id of the first person in that record, and back from a program id to the Fisher id of one record where that person is listed. There are two central tables in the database – the person table and the relationship table.

The person table (person_table) has information exclusive to the individuals. The columns of this table are : Program id, Name , Birth date, Death date, Address, Gender, Special status, Number of spouses, and Other Information. The Special Status field contains information about whether a person is a twin, triplet, foster child, or adopted child. The relationship table (child_parent_table) keeps information relating the individuals. It contains an entry for each couple and a list of their children, if any, in terms of their program id specified in person_table. The columns of child_parent_table are: Program id of father, Program id of mother, Marriage date, Program id of each child. The last field is a list of program ids of each child, if the couple has any. As each row/column combination can have only one value, the list is kept as a string with special delimiters between children.

PedHunter contains a utility program for creating generation_table that contains the maximum generation level for each person relative to other persons in the database. The two columns of this table are program id of the individual and maximum generation level. The generation_table is used in computations including inbreeding and kinship coefficients.

Graphs and related terminology

A *graph* is a mathematical representation for objects and relationship between pairs of objects. Some standard graph terminology is defined below and illustrated by references to the graph in Figure 1. The set of objects represented in a graph are called its *vertices*. If two objects are related, they are *joined* by an *edge*. For convenience, the following notation is used:

- $V(G)$ represents the set of vertices for graph G.
- $E(G)$ represents the set of edges for graph G.
- e is $u — v$ if edge e joins vertices u and v and the *order* of u and v is not important. Such an edge is called an **undirected** edge.
- e is $u \rightarrow v$ if edge e joins vertices u and v and the edge is ordered *from* u *to* v. Such an edge is called a **directed** edge.
- $v \in V(G)$ if v is a vertex in graph G.
- $e \in E(G)$ if e is an edge in graph G.

A graph is a *directed* graph if its edges are directed edges; otherwise, it is an *undirected* graph. A pedigree can be represented by a graph that has a vertex for each person and an edge for each child-parent pair among the persons in the pedigree. Sometimes it is desirable to make the graph directed by directing each edge from the vertex for parent to the vertex for child. For example, the pedigree (Figure 2) that is used later as an example, can be represented as in Figure 1. It is useful to view a pedigree as a graph because it traces the passage of genetic information over generations and provides a convenient mathematical representation. A graph representing a pedigree is called a *pedigree graph*. For the pedigree graph in Figure 1, the set of vertices $V(G)$ is $\{1, 2, \ldots, 32\}$ and there are 32 directed edges including $6 \rightarrow 7$ and $17 \rightarrow 23$.

A *path* from vertex u to vertex v in G is an alternating sequence of vertices and edges of G, beginning with u and ending with v, such that no vertex is repeated and every edge joins the vertices immediately preceding it and following it. A path p connecting u to v is denoted $p = u, u_1, u_2, \ldots, u_n, v$ where $u_1, \ldots, u_n$ is the sequence of other vertices on the path. For example, $1 \rightarrow 3 \rightarrow 4 \rightarrow 5$ is a directed path in the graph of Figure 1. The undirected sequence $14 — 13 — 12 — 11 — 2 — 3 — 4 — 5$ is a path in the undirected version of this graph, but not in the directed version. The two paths are denoted as $p = 1, 3, 4, 5$ and $p = 14, 13, 12, 11, 2, 3, 4, 5$, respectively.

A directed path p from u to v in a pedigree graph traces one way v receives genetic material from u. The vertices on p is also a (partial) list of ancestors of v. Absence of a directed path from u to v indicates that u is not an ancestor of v. Hence, there is no way that v can receive genetic material from u; e.g, in Figure 1, person 28 does not receive genetic material directly from person 6, although they have a common ancestor.

A pedigree graph G *connects* a set of individuals S if there is a vertex in $V(G)$ for each individual in S and in the *undirected* version of the graph, there is a path between each pair of vertices representing people in S. The pedigree of Figure 1 is *connected*. Geneticists usually use the term *pedigree* to mean a set of parent-child relationships that induce a pedigree graph whose undirected version is *connected*. Therefore, any formal definition of *pedigree construction* should require that the output pedigree induce a connected pedigree graph. However, this requirement does not specify the output pedigree uniquely, and different specifications of pedigree construction are useful for different purposes.

A *cycle* is a path that begins and ends at the same vertex. If a graph has no cycles, it is called *acyclic*. Any directed pedigree graph is acyclic since the edges are directed from parent to child, and it is biologically impossible to have a cycle in this setting. The term "loop" is not used here because formally, a *loop* in a pedigree is a cycle in a more complicated, undirected representation called a *marriage graph*(Lange and Elston 1975). For example, a first-cousin marriage with offspring leads to a loop in the undirected marriage graph, but not to a cycle in the directed pedigree graph.

The *length* of a path is the number of edges in the sequence defining the path; e.g., the length of path $p = u_0, u_1, \ldots, u_n$ is n. A path from vertex u to vertex v is a *shortest path* if it has minimum length of any path from u to v. The length of a shortest path from u to v is also called the *distance* $d(u, v)$ from u to v.

A shortest path from u to v makes the *fewest assumptions* about how an allele could have passed from u to v. By the principle of parsimony, a hypothesis that a disease-causing allele passed from u to v, requires consideration of *all* shortest paths from u to v. This observation motivates the *asp* pedigree construction query described below.

If G and H are graphs then H is a *subgraph* of G if and only if $V(H)$ is a subset of $V(G)$ and $E(H)$ is a subset of $E(G)$.

Given a pedigree graph G that connects a set of people S, it is sometimes useful to find a subgraph of G that still connects S. One such situation is when the pedigree graph G has too many edges to be drawn comprehensibly. A subgraph of G that contains minimum number of vertices and connects S is called a *minimal* subgraph, and the problem of finding a minimal connected subgraph is called a *Steiner tree* problem in the computer science literature. Therefore a minimal pedigree subgraph that connects specified individuals is called a *Steiner pedigree*. PedHunter finds Steiner pedigrees with the query called *minped* listed below. Steiner pedigrees are useful for drawing and for linkage analysis when the pedigree graph has too many edges. PedHunter's method to construct Steiner pedigrees and some background from the computer science literature are explained in the next section.

Steiner pedigrees

Steiner tree problems are widely studied due to applications such as designing road networks and VLSI chips. A survey of Steiner tree problems can be found in (Hwang and Richards 1992). Many Steiner tree problems are quite intractable computationally, but pedigree graphs have special features that may make the problem easier. Pedigree graphs are acyclic and the vertices that must be included typically represent children at the bottom generation. Such vertices that have no outgoing edges are called *terminals*. In a recent paper Zelikovsky (Zelikovsky 1997) studied the Steiner tree problem for all directed acyclic graphs with the constraint that the vertices to be connected are terminals. This problem has applications outside genetics. Zelikovsky showed that this version of the Steiner tree problem is intractable in a formal sense, and gave an approximation algorithm for the problem. An *approximation algorithm* finds a subgraph that contains all the prescribed vertices whose size is bounded relative to the size of the minimal subgraph where the **size** of a graph is the number of vertices in the graph. However, both the performance ratio and the computation time of this approximation algorithm grow exponentially with the length of the longest path in the graph.

Instead of Zelikovsky's approximation algorithm, PedHunter uses an algorithm that finds an exact smallest subgraph (subpedigree) as measured by the number of vertices (individuals). The approximation algorithm may be inadequate for the subsequent linkage analysis. For the typical problem instances arising on pedigree graphs, requires reasonable computer running time.

The exact algorithm applies a general technique called *branch and bound* that is commonly used to explore exponential-size search spaces in intractable search problems (Papadimitriou and Steiglitz 1982). Branch and bound works well in problems where one can quickly get to a good solution. The weight of the known solution can be used to eliminate many partial solutions whose weight is larger (for a minimization problem such as Steiner tree). When a partial solution is eliminated, all full solutions containing that partial solution are implicitly eliminated also. "Branch" refers to steps that expand the current partial solution by branching in the search space; "bound" refers to using a known solution as a bound.

Branch and bound algorithms start by finding an initial solution, keep track of the current best solution and probe other solutions until it is possible to determine that the current solution is at least as good as any other solution. Branch and bound algorithms typically use *retracing* or *backtracking* which means that part of the current (partial solution) is removed, and the algorithms proceeds to consider other solutions that do not contain the removed part. To describe a specific branch and bound algorithm one must describe:

1. What makes one solution better than another ?
2. How to find an initial solution ?
3. How to probe all solutions?

For the minimal pedigree problem, the algorithm solves the three basic questions as follows.

Comparing solutions. The number of edges in the solution is the measure for comparing solutions and the solution with fewer number of edges is better. The number of edges in a solution is called the *cost* of the solution. The problem statement asks to minimize the number of vertices, but any connected graph with no cycles that has e edges, must have exactly $e+1$ vertices [8]. Therefore, minimizing number of edges is equivalent to minimizing number of vertices. The algorithm transforms the initial Steiner problem where all edges have weight 1 to a related problem where edges may have larger weight. In the weighted version, the cost of a solution is the sum of weights of edges the solution contains.

Initial solution. The steps for finding an initial solution are as follows:

1. Start with the specified set of individuals as the vertices in the solution.

2. For each vertex v in the solution, except the root, that is not connected to a parent, choose a parent p that is present in the input graph. If the parent chosen is not in the solution, add a vertex for p in the solution. Add the edge $p \to v$ to the solution.

3. Repeat Step 2 until there is only one vertex in the solution that is not connected to a parent. That vertex is the root of the pedigree.

Probing all solutions. Every edge in the input graph G is either *in* the current solution or it is *out*. The edges are numbered $1, \ldots, |E(G)|$ such that edge $u \to v$ is preceded in the order by all edges that are incoming into u. Such an order is possible for pedigree graphs because they are acyclic [8].

Consider the edges in order $e = 1, 2, 3, \ldots$. Add edge e to the current solution unless it (i) creates an undirected cycle or (ii) creates a (partial) solution with higher cost than the current best. When the current partial solution is a full solution (detectable by testing whether number of edges equals number of vertices minus 1), compare its cost to the current best.

There are four cases that need to be specified further.

1. If the current partial solution is a full solution and it has lower cost than the previous best, replace the best solution and retrace decisions back to the last edge added *in*. Omit that edge and proceed to look for better solutions.

2. If the addition of e creates an undirected cycle, then omit e and proceed to add more edges.

3. If the addition of e creates a partial solution with higher cost than the current best full solution, then as in case 1, retrace back to the last edge added *in*. Omit that edge and proceed forward.

4. If the last edge was considered, and the solution is not full, then as in case 1, backtrack to the last edge added *in*, omit it and proceed.

All solutions have been considered when retracing returns all the way past edge 1.

The worst-case computation time of the algorithm is exponential in the number of edges in the input graph. The computation time is lowered substantially by reducing the size of the graph to be considered in the branch and bound algorithm. The graph is reduced by three transformations that are applied repeatedly in any order until none of them can be applied:

1. Replace paths consisting of $k+1$ vertices with 1 incoming edge and 1 outgoing edge with a single edge of cost k. This is valid because in the original graph, either all k edges must be included or excluded from a solution.

2. Delete unselected terminals with 1 incoming edge. This is valid because they cannot serve to connect the selected terminals.

3. Delete edge $u \rightarrow v$ if there exists another path from u to v. This is valid because the input graph is an all shortest paths pedigree. Therefore, all paths from u to v have the same cost. The vertices on the multiple edge path cannot lead to a worse solution than using the single edge $u \rightarrow v$ and might lead to a better solution.

The first transformation introduces edges whose weight is > 1. The third transformation can apply only after the first transformation has created edges of weight > 1 because the cost of all paths from u to v is the same. The cost of a solution in the transformed graph is the sum of costs of all edges in the solution. The transformation is reversed to produce the output Steiner pedigree.

Figure 1: Graph for the partial McKusick pedigree

Figure 2: Partial McKusick pedigree (MKped)

Figure 3: All shortest paths pedigree (ASPped)

Figure 4: Minimal (Steiner) pedigree