Efficiently Querying Contradictory and Uncertain Genealogical Data

Lars E. Olson
David W. Embley
Department of Computer Science
Brigham Young University
Provo, UT 84602
{olsonle,embley}@cs.byu.edu

1. Introduction

In doing genealogical research, it is common to find multiple sources of information that provide contradictory or uncertain data. Often, users cannot resolve the conflicting data and wish to keep all possible values, and may also wish to add degrees of certainty to the uncertain data so that the most probable interpretations can be given. Most simple databases require the conflicts and uncertainty to be resolved, and are therefore not well suited for genealogy data.

As an example, suppose we are collecting genealogical information, and for a particular person we find two possible dates of birth (such as "4 or 5 December 1394") or an imprecise date of birth (such as "December 1394," with a month and year but no day). There may be no way of ever finding out the true date of birth, although we may be more confident in one date than another. Since we typically want to have as much identifying and certainty information as possible, the database should store as much as it can, even though doing so may violate a constraint.

Given that genealogical data is contradictory and uncertain, two interesting questions regarding queries immediately arise. (1) What is the complexity of querying the data? (2) How do we find a valid subset of the data that is consistent with the constraints, and if more than one such subset exists, how can we efficiently determine which is most likely? In Sections 2, 3, and 4, we give brief answers to these questions. An in-depth study is under way [O02] to verify the correctness of these answers.

2. Conflicting and Uncertain Data

Databases that allow a disjunction (the "OR") of values in place of single values are called *disjunctive databases*. Some theoretical foundations and proposed models of disjunctive databases already exist (see for example [IV89], [AG85], and [KW85]). It has been proven in [IV89], however,

This material is based upon work supported by the National Science Foundation under Grant #0083127

that queries on disjunctive databases in general have CoNP-complete time complexity (i.e. all known algorithms require an exponential amount of time based on the problem size). Thus, using an unrestricted disjunctive database makes querying the data intractable. Based on a theorem presented in [LYY95], however, we present a way to handle genealogical data so that many queries, if not most, become tractable. Furthermore, the theorem also gives us a way to determine which queries remain intractable, which gives us the opportunity to handle these queries heuristically (i.e. using a quick estimate) or under reasonably bounded extents.

The idea of the theorem is based on the notion of disjunctive graphs; that is, graphs with hyperarcs that represent disjunctions. Figure 1 shows an example of a disjunctive graph, along with one of its possible interpretations (where each hyperarc is replaced by one of the possible arcs it can represent). Disjunctions can occur on the head side of the arc (such as the arc from *a* to {*b*,*c*}) or on the tail side (such as the arc from {*c*,*d*} to *f*) or on both sides. Since only one of the disjunctive heads or tails can hold, each disjunction gives rise to a different interpretation. There are multiplicatively many interpretations, which is why disjunctive data naturally leads to intractability.

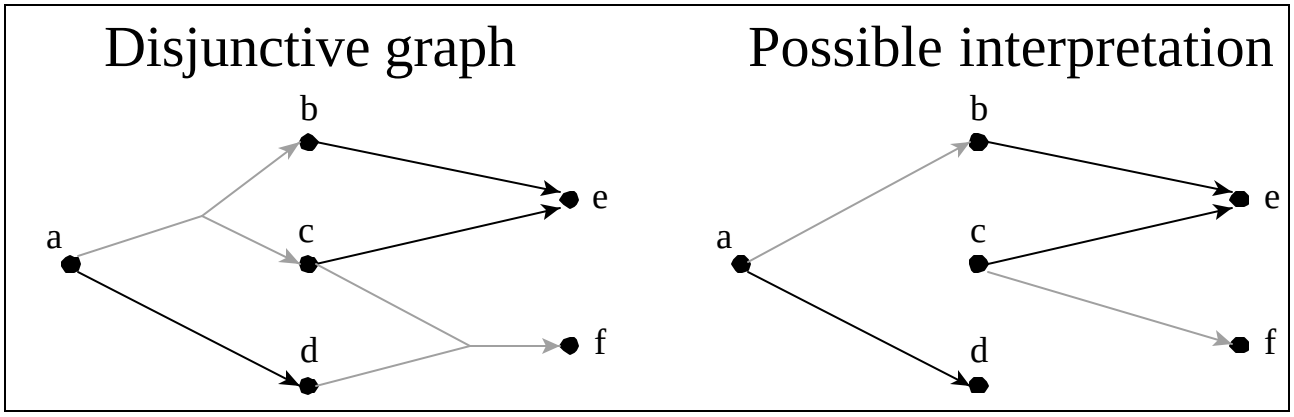

Figure 1: An example of a disjunctive graph (left) containing two disjunctive arcs, and one of the four possible interpretations of the graph (right).

[LYY95] considers the problem of computing the transitive closure of a node *x* in a disjunctive graph, which is defined as the set of all nodes *y* such that in every possible interpretation of the disjunctive graph, there exists a path from *x* to *y*. For the graph shown in Figure 1, for example, the transitive closure of node *a* is the set {*a*, *d*, *e*}. The theorem states that if each disjunctive arc of the graph contains a disjunction only in the head of the arc (rather than in the tail), computing the transitive closure is a polynomial-time algorithm; otherwise, it is CoNP-complete.

To provide for uncertainty, we use weighted disjunctive graphs. We can place weights on disjunctive heads or tails. Thus, for example, if we place 0.7 on the head pointing to *b* in Figure 1, we are saying that we believe with 70% confidence that (*a*,*b*) holds. Heads and tails without weights may be assigned default weights.

Table *Person*:

ID#	Name	Birth Date	Birth Place ID#	Marriage Date
1	John Doe	12 Mar. 1840 or 12 Mar. 1841	1 or 2 (0.8)	15 Jun. 1869 (0.5) or 16 Jun. 1869 (0.7)
⋮	⋮	⋮	⋮	⋮

Table *Place*:

ID#	City	State
1	Commerce or Nauvoo	Illinois
2	Quincy	Illinois
⋮	⋮	⋮

Figure 2: Genealogy database to be converted to a disjunctive graph, including certainty measures for some values. Note that attribute Birth Place ID# is a foreign key referencing table *Place*.

To see how to map conflicting and uncertain data to disjunctive graphs, consider the disjunctive database in Figure 2. We can transform this data into a disjunctive graph by drawing arcs from each of the key values to their corresponding attributes, using disjunctive arcs where necessary. We also add arcs representing foreign keys by drawing arcs to the actual nodes in the table being referenced. We can attach labels to each arc to represent the attributes. Figure 3 shows the transformed graph.

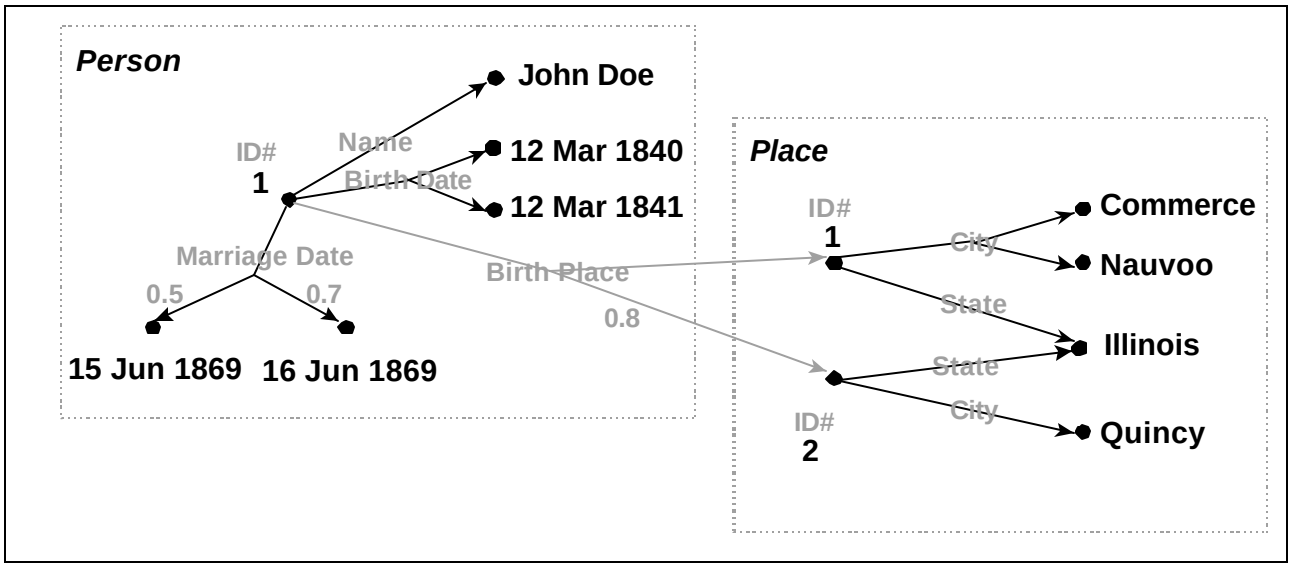

Figure 3: Database from Figure 2 transformed into a disjunctive graph.

3. Query Processing with Disjunctive Graphs

Consider the query, "In what state was the person with ID#1 born?" (which can be written as $\pi_{State}(\sigma_{ID=1}\text{Person} \bowtie \text{Place})$ in relational algebra notation.) To answer this query, we compute the transitive closure of the node labeled ID#1 and return the node corresponding to the attribute "State," which in this case is Illinois. We can guarantee that no arcs with disjunctive tails will appear for a database such as this if we insist that no disjunctions for primary-key attributes occur (e.g. the relation will never contain "ID#1 or ID#2" for a single object) and that foreign keys only reference primary-key attributes in other tables. Since the only arcs created in the transformation originate from these object identifiers, we will never have a disjunctive tail, and therefore this type of query can be answered in polynomial time.

Not all queries can be answered this easily, however. If we consider the query, "In what city and state was the person with ID#1 born?" (which can be written as $\pi_{City,State}(\sigma_{ID=1}\text{Person} \bowtie \text{Place})$), we still return Illinois as the state, but we find no city in the closure (as [LYY95] defines it). The correct response to this query depends on what the user really means in the query, which could be one of three desired answers:

- What values do we know without a doubt?
- What are all the possible values for each attribute?
- What are the most likely values?

For the first question, since we do have doubt about the correct city, the correct response is to return nothing for the city and Illinois for the state, and thus the transitive closure does indeed give the correct solution. For the second question, it should return Commerce, Nauvoo, and Quincy as possible cities and Illinois as the state. To return this answer, we perform another simple transformation on the graph by replacing all disjunctive arcs with regular arcs to all the possible attributes, as Figure 4 shows. This graph becomes a degenerate case in which there are no disjunctions, and thus computing the closure is still in polynomial time. Section 4 discusses the complexity of answering the third question.

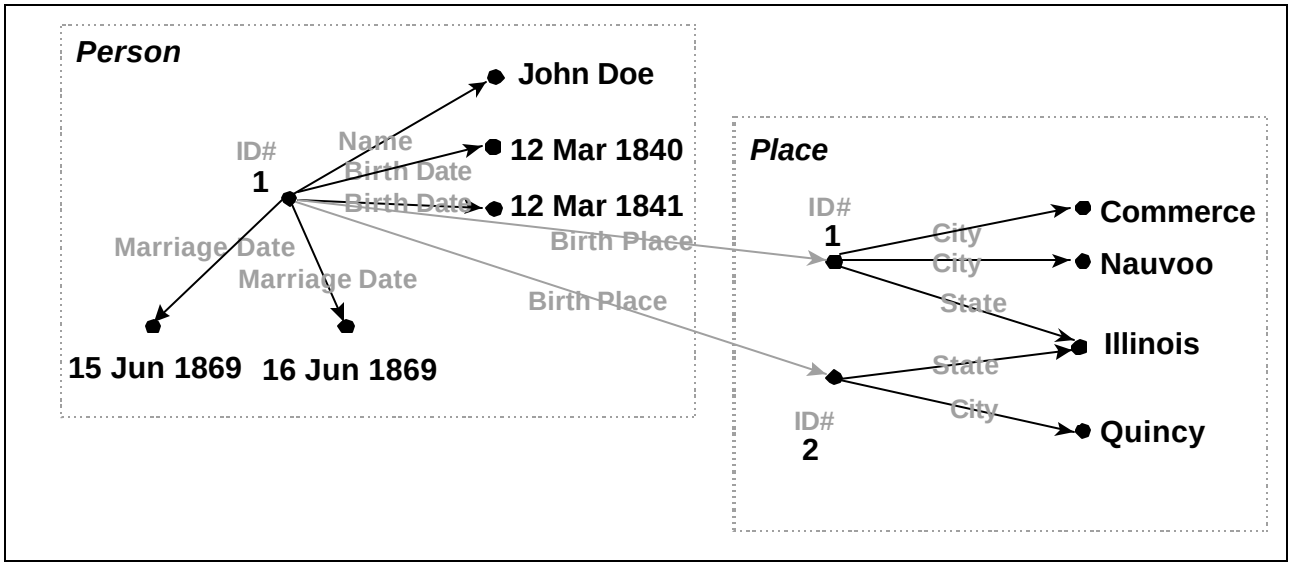

Figure 4: Graph from Figure 3 transformed to consider all possible values as true. The certainty measures have been omitted.

When a query requires selection on non-key attributes, such as, "Find the names of all people born in Nauvoo between 1841 and 1844" (which can be written as $\pi_{Name}\,\sigma_{BirthDate \geq 1841 \wedge BirthDate \leq 1844 \wedge City = Nauvoo}$ (Person $\bowtie$ Place)), rather than on a key attribute, such as the queries already discussed, we can still guarantee polynomial running time. We compute the closure for every possible ID# (which is bounded by the number of nodes n in the graph, and thus the running time is bounded by $n * P(n)$ where $P(n)$ represents the time required to compute the closure for one node). For each ID#, if the Date falls within the specified range and the City attribute is "Nauvoo," we add the Name attribute to the answer. Again, this answer will vary depending on whether the user wants what is definitely known, what all the possible answers are, or what the most likely answer is.

Most genealogical queries can be handled in this manner to achieve polynomial running time, but there are some exceptions. As long as tables are appropriately joined on object identifiers, the disjunctions will always be in the heads of the arcs, but some queries require joins on other attributes. Consider, for

example, the query, "Which people have the same birth date?" (which can be written as $\pi_{P1.Name,\ P2.Name}$ (Person P1 $\bowtie_{P1.BirthDate\ =\ P2.BirthDate}$ Person P2).) A portion of the graph needed to solve this query might look like Figure 5. Notice that to answer this query, since the join attribute is not the key attribute, we must add disjunctive arcs from the join attributes to the corresponding object identifiers. This creates arcs with disjunctions in the tail, and cannot therefore be solved using a polynomial-time algorithm. Again, the correct response to this query depends on whether the user wants what is definitely known, what all the possible answers are, or what the most likely answer is. If the user wants what is definitely known, we can offer partial answers by simply removing all the disjunctive arcs entirely, or by asking the user to limit the search space. If a partial answer is not acceptable, we can at least detect when such a query is CoNP-complete, warn the user if the size of the graph is large, and ask how we should proceed. If the user wants an enumeration of all the data for each attribute, we can make the same transformation we performed in Figure 4, and the query will be polynomial.

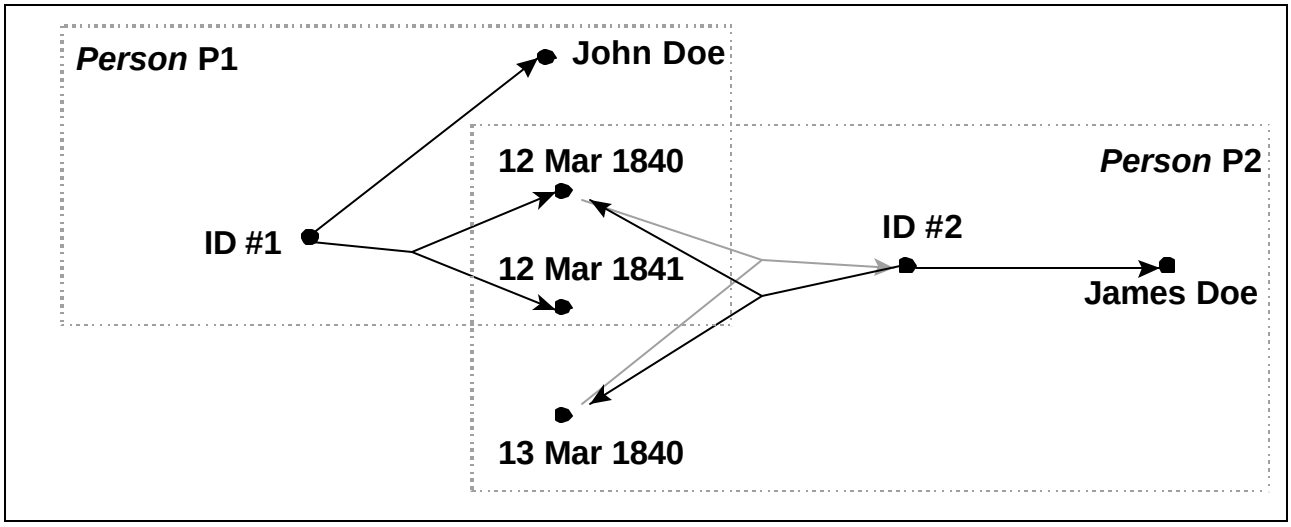

Figure 5: Query including a join on a non-key attribute.

4. Uncertainty in Disjunctive Graphs

In general, determining which values are the most likely, based on certainty measures and given constraints, can be a difficult problem to solve. In our particular application of genealogy, most disjunctions appear to be mutually independent. Determining whether John Doe's birth date was in 1847 or 1848, for example, probably will not affect the choice of whether his great-grandfather's name was Joshua or Jacob. Disjunctions that are not mutually independent can usually be limited to immediate family relations. For example, determining the correct birth dates of a grandchild and a grandparent can be decided by comparing the possible birth dates of the grandchild with the grandchild's parents (an

immediate family relation) and the grandchild's parents with the grandparents (also immediate family relations). Thus we can usually limit our search space to parents, siblings, and children. It is important to note that limiting the search space will only yield locally optimal answers, which could be the same as (or very close to) the globally optimal answer, but cannot be guaranteed to be so.

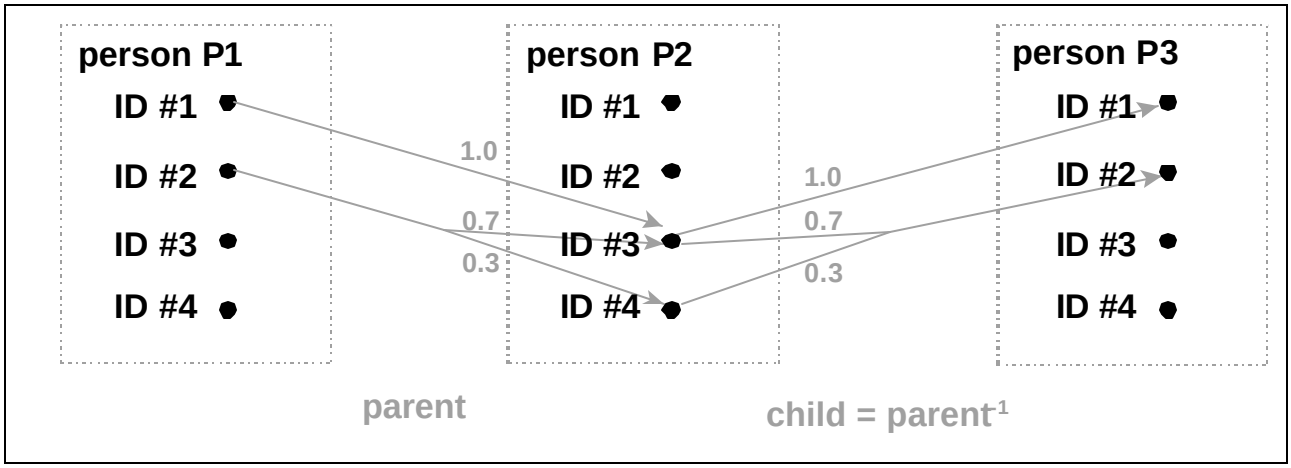

Figure 6: Graph creating all immediate family connections, including certainty measures. Other attributes, such as Birth Date, have been omitted to keep the diagram simple.

Assuming there are n values in the database, each arc in the disjunctive graph can have a branching factor of k, which is at most (and probably considerably less than) n. We can make connections between all the people in the database with their parents in a graph of depth 2. We can make connections with their siblings by inverting the connections on the parent graph to find all the children. The resulting graph has a depth of 3, as Figure 6 shows. Since each person appears once at each depth level, there are at most n arcs between nodes of depth 1 and nodes of depth 2. Similarly, there are at most n arcs between depth 2 and depth 3. Since each arc has a branching factor of k, we have at most nk possible choices between depth 1 and depth 2, and another nk between depth 2 and depth 3. The total possible choices are n^2k^2, which is bounded by n^4. Thus, a brute-force backtracking algorithm to calculate the likelihood of each possibility (and pick the most likely choices), using standard techniques for processing uncertainty such as those explained in [GC97], can be done in polynomial time.

5. Conclusion

In a genealogy application, it is possible to model disjunctive data in such a way that most queries on the data only require polynomial execution time. Some queries are still intractable, but we can detect these queries and either execute them when the problem size is reasonably small, or otherwise offer the user heuristics to give partial answers and locally optimal answers.

Bibliography

[AG85] S. Abiteboul and G. Grahne, "Update Semantics for Incomplete Databases", *Proceedings of the 11th International Conference on Very Large Databases (VLDB)*, Aug. 21-23, 1985, Stockholm, Sweden, pp. 1-12.

[GC97] N. van Gyseghem and R. de Caluwe, "The UFO Database Model: Dealing with Imperfect Information", *Fuzzy and Uncertain Object-Oriented Databases: Concepts and Models*, ed. R. de Caluwe, World Scientific Publishing Co. Pte. Ltd., 1997, pp. 123-185.

[IV89] T. Imielinski and K. Vadaparty. "Complexity of Query Processing in Databases with OR-Objects," *Proceedings of the Eighth ACM Symposium on Principles of Database Systems (PODS)*, March 29-31, 1989, Philadelphia, Pennsylvania, pp. 51-65.

[KW85] A. M. Keller and M. W. Wilkins, "On the Use of an Extended Relational Model to Handle Changing Incomplete Information", *IEEE Transactions on Software Engineering*, Vol. 11, No. 7, July 1985, pp. 620-633.

[LYY95] J. Lobo, Q. Yang, C. Yu, G. Wang, and T. Pham, "Dynamic Maintenance of the Transitive Closure in Disjunctive Graphs," *Annals of Mathematics and Artificial Intelligence*, Vol. 14, 1995, pp. 151-176.

[O02] L. Olson, "Permitting Constraint Violations in Data Storage for Integrated Data Repositories," Master's Thesis, 2002, in progress.