ECE356: Database Systems Winter 2018

Lecture 35 — Recovery: Repair, Probability

Jeff Zarnett

Probabilistically Answering Queries

Residues and repairs are not the only way to return consistent answers. When there are several options, as in Table 1, we can examine these options and make a determination of which is more likely. We form candidates - pretenders to the throne of the correct database - by breaking up the possibilities for repair into all of their possible variants; in each candidate database, one tuple from each cluster is selected. We see that in the example of Table 1, both candidate databases will receive a probability of being the "correct" database.

salaries	employee_name	salary
	J. Page	50 000
	J. Page	80 000
	V. Smith	35 000
	M. Stowe	75 000

Table 1: Inconsistent Salaries Table [BC99]

Finding Consistent Answers Andritsos, Fuxman, and Miller in [AFM06] take the approach of examining various probabilities of each tuple in a cluster being the correct answer. In the simplest solution, the database will just answer queries and attach the probability of the answer's correctness as another attribute of the tuple. Referring back to the salary example, below in Table 2 is Table 1 modified with the respective probabilities of each item:

salaries	employee_name	salary	probability
	J. Page	50 000	0.1
	J. Page	80 000	0.9
	V. Smith	35 000	0.4
	M. Stowe	75 000	1

Table 2: Inconsistent Salaries Table with Probabilities [BC99] [AFM06]

If the query being asked were all the names of all employees making more than $70 000, a probabilistic assessment would be performed to decide what certainty we could indicate the answers with. In the trivial case, M. Stowe's salary is certainly greater than $70 000, since its probability is 1 (completely certain). Uncertainty enters the picture when examining the J. Page tuples. We note probability of his salary being $80 000 is 0.9, so we include him in the return set and indicate the attached probability. If the query were names of employees making more than $45 000, then we would return J. Page with probability 1, since the sum of the probabilities of the tuples wherein his salary exceeds $45 000 is 1.

Andritsos, Fuxman, and Miller simply modify the requested queries to include the probability attributes. If the original query was:

```
SELECT s.employee_name FROM salaries s WHERE s.salary > 70 000
```

then the only changes are the addition of `SUM(s.probability)` and `GROUP BY s.employee_name`, so that the re-built query reads:

```
SELECT s.employee_name, SUM(s.probability) FROM salaries s WHERE s.salary > 70 000
GROUP BY s.employee_name
```

In a more complex query, we simply multiply the probabilities. Consider the following query [AFM06]:

`SELECT o.id, c.id FROM order o, customer c WHERE o.cIdFk = c.id AND c.balance > 10000`
This query is modified in the same way as that of the preceding paragraph, except the sum statement reads `SUM(o.probability * c.probability)` and the group statement is `GROUP BY o.id, c.id`.

Determining Probabilities The most pressing question is how we determine the probabilities. We note there are definite conditions attached to being able to determine these. See Figure 1 for the detailed breakdown of the procedure to map the tuples' probability to the interval [0, 1].

Input : A set of tuples $\mathbf{T}$,
- a clustering $\mathcal{C} = \{c_1, c_2, \ldots, c_k\}$ of $\mathbf{T}$,
  where c_i is the identifier of cluster i
- a distance measure d.

Output : For every tuple $\mathbf{t}$ in $\mathbf{T}$, a probability $prob(\mathbf{t})$.
Main Procedure :
- (Step 1) For $i = 1 \ldots k$:
  * compute cluster representative rep_i for c_i
    by merging all the tuples that belong to it.
  * initialize sum of distances for c_i, $S(c_i) = 0$.
- (Step 2) For each tuple $\mathbf{t} \in \mathbf{T}$ that belongs to c_i:
  * compute $d_{\mathbf{t}} = d(\mathbf{t}, rep_i)$, the distance of $\mathbf{t}$
    to the representative of its cluster.
  * Add $d_{\mathbf{t}}$ to $S(c_i)$.
- (Step 3) For each tuple $\mathbf{t} \in \mathbf{T}$ that belongs to c_i:
  * compute similarity $s_{\mathbf{t}} = 1 - \frac{d_{\mathbf{t}}}{S(c_i)}$.
  * $prob(\mathbf{t}) = 1.0$ if $|c_i| = 1$, or
    $prob(\mathbf{t}) = \frac{s_{\mathbf{t}}}{|c_i|-1}$ otherwise.

Figure 1: Tuple Probability Assignment Formula [AFM06]

Though most of the algorithm is self-explanatory, some components merit clarification, also provided in [AFM06]. Tuples within a cluster must be exclusive events, or the algorithm will fail. Representatives are determined by commonality. Representatives contain the most common features in the cluster, so the similarity between the tuples will give an indication of which tuple is most likely to be representative. To clarify, a short example, Table 3:

customers	name	market_segment	country	address
	Mary	building	USA	123 Jones Ave.
	Mary	banking	Canada	123 Jones Ave.
	Marion	banking	USA	123 Jones Ave.

Table 3: Inconsistent Customers Table [AFM06]

When examining this with a human's eyes, we might conclude that the most common values in the database are probably the correct ones, so the resultant representative tuple would show the customer Mary as part of banking in the USA with an address of 123 Jones Ave. The algorithm shown in Figure 1 follows this same intuition. The tuple we will consider correct is the one closest to the representative. For numerical data, similarity between two figures can be computed, and a pair can be more or less similar than another pair of figures (456 and 385 are more similar than 750 and 385). For data for which there is no obvious distance measure, we term them *categorical data*, and we proceed using information loss as the distance metric [AFM06]. Information loss is simply a measure of the difference between the tuple in question and the representative.

Analytical Shortcomings The strategy presented is imperfect, however, since we might fail to produce clean answers for some classes of query. Andritsos, Fuxman, and Miller present in [AFM06] the following case as unable to succeed: `select c.id from order o, customer c where o.quantity < 5 and o.cIdFk = c.id and c.balance > 25 000`. This fails because the join between `c` and `o` incorrectly double-counts some probabilities. Obviously, query rewriting cannot succeed for all cases, so the algorithm is limited to only those which we

can reliably rewrite. Andritsos, Fuxman, and Miller argue – without proof – that such un-rewritable queries occur only infrequently, and that the algorithm is still valid most of the time.

Overlooked in this analysis is determining how we might find a representative sample when it is not obviously in a majority-rules scenario. Given two prima facie equally plausible tuples, how do we decide which is the better choice to consider the representative? Whichever is picked to be the representative will not differ from the representative (by definition), so it will receive a probability of 1. Thus, whatever we (perhaps randomly) choose to be our representative is the eventual winner and will be considered correct. It is clear that we need to come to some decision, but reporting 100% certainty about data which is, at best, 50% certain is misleading.

This analysis also equates popular with correct. If an incorrect answer appears in the database twice and a correct answer once, then the incorrect answer will be chosen and deemed correct, since it is more popular. That aside, there is not much that can be done about this problem; even a human observing the database might be more likely to conclude that the popular answer is the correct one, in the absence of additional, external knowledge.

Computational Complexity of Repair & Probability

Computational complexity is broken down into a short analysis on each of the methods detailed above. Each approach has its own properties. Computational complexity may disqualify a method from being practically useful, should it take too much time to reach a reasonable answer.

Repairing Databases Assuming that we are looking at repairs that are subsets of the original database, then repair checking is in polynomial time, for arbitrary constraints combined with acyclic dependencies [BC99]. Should any of these constraints not hold, the problem is pushed into the realm of co-NP-complete problems, although additional orthogonal restrictions might lead to more polynomial time cases [BC99].

Having considered the spectre of computationally infeasible or impossible situations, Bertossi and Chomicki [BC99] develop various methods of finding consistent answers without explicitly computing every possible repair to the database. One of the ways that they suggest is compact repair representation - use information present to construct an efficient representation of all the possible repairs, and use this representation to answer all queries. However, this is not explored in the paper.

Query Transformation The process of query transformation in [BC99] is shown to have a polynomial time computability of result tuples, as the transformed query will be first order as long as the original query is as well. In fact, because the query transformation does not require examining all the possible repairs, so we can evaluate a query with an exponential number of possible repairs in polynomial time.

Aggregate Query Transformation The approach presented in [BC99] is to aggregate queries is to build a conflict graph; a standard graph with nodes and edges. In the graph, maximal independent sets - that is, sets that are the farthest apart in terms of data equality - represent possible repairs of the database. As long as there is at most one nontrivial constraint, all operators except `COUNT` are polynomial time. If there are many nontrivial constraints, then the problem of finding lower and upper bounds becomes NP-complete. The `COUNT` operation is always NP-complete, even in the case of a single nontrivial constraint. Finally, it is noted that the `AVG` function's polynomial time algorithm is iterative, and cannot be formulated using SQL.

Under special circumstances, Bertossi and Chomicki found some better behaviour for the `COUNT` function, but even approximating is difficult; the concept of a maximal independent set reportedly has bad approximation properties.

Probabilistic Complexity Andritsos, Fuxman, and Miller include in [AFM06] a note on the computational complexity of their work. specifically a graph showing the performance as the size of the database grows, reproduced here as Figure 2:

It is clear from a cursory examination that the behaviour of the queries is linear, with the exception of some corner cases (like Query 3). However, they sidestep the issue that the number of candidate databases is possibly exponential - if there are n inconsistencies each with clusters of size m, the total possibilities might be as large as

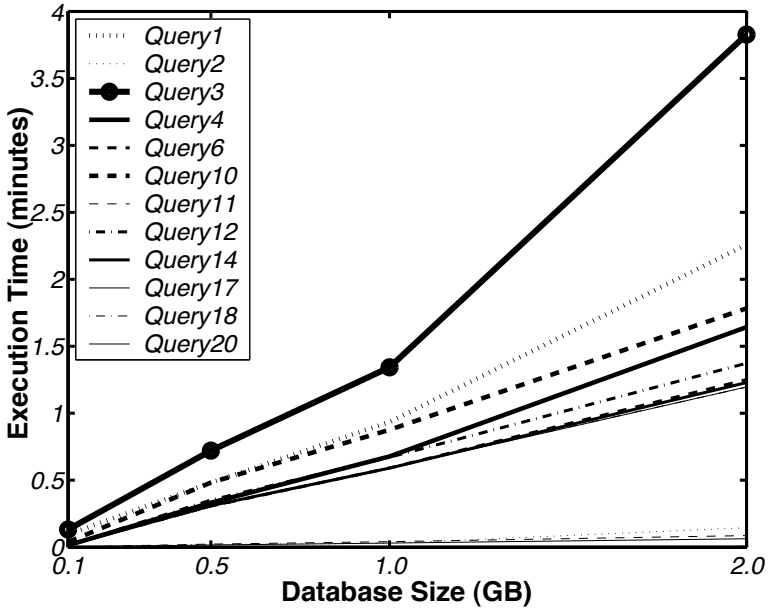

Figure 2: The Effect of Database Size on Probabilistic Evaluation [AFM06]

n^m. Computing probabilities for such a set might be exceedingly difficult, but the whole issue is glossed over with a single comment.

Future Work & Suggestions

In [M. 99], Arenas, Bertossi, and Chomicki warn that their proofs of termination and completeness properties of the query transformation are only preliminary and partial. Surely this will be addressed in the future. They also find that completeness for disjunctive and and existential queries, they will need to move beyond simple query transformations. In addition, they do not yet have upper bounds on the size of the transformed query, and they lack some complexity information on different classes of query. This particular paper ([M. 99]) is a more detailed examination of some sections of [BC99]), with two authors in common, so suggestions for improving it are combined with that of its successor.

Database repairs examined in the [BC99] fashion are based on differences against the whole tuple and does not permit modifying attributes of tuples. Bertossi and Chomicki suggest that a *flexible* (attribute-level) repairs. They further identify majority-based approaches to consistency, which is very similar to how the probabilistic approach decides what to do.

Further in that paper, the section on logic programs to compute database repairs is rather light on details and seems mostly an aggregation of examples. We suggest that the paper might be improved by including more detail on the subject, since a lot has been left out, or reducing the amount of space dedicated to what is clearly a side point implementation detail.

In this same vein, there is a note about computationally constructing compact repair representations, yet there is not much talk about how we might do this, aside from a side note about placing `null` into a field when there are infeasibly many repairs. We suggest expanding compact repair representations, or clarifying the note about them (which implies that there is a wealth of information on the topic coming).

The probabilistic paper presents no future extensions or future directions to its work. As suggestions for them, they could certainly investigate the computational complexity further. Specifically, why some cases are more difficult than others to handle (see query 3 in Figure 2). They also appear to lack a strategy for dealing with a large set of inconsistencies.

Noting the scenario in which 50% certain data may be reported 100% certain, my suggestion for such forced-guess scenarios is to report the uncertainty associated with this answer and confess that we cannot find any better solution without additional input.

Finally, the set of queries that they cannot rewrite is claimed to be small, but no proof or evidence is offered to convince the reader that this is really the case. Surely a subsection or followup would be rather more persuasive than simply taking it on faith.

Conclusion

There appears to be no single right answer when dealing with an inconsistent database. However inconsistent it may be, it is unlikely that we will be able to unequivocally say what is correct and what is not. All of the examined options present viable alternatives to solving the problem, but they can easily slip into computational infeasibility in the event of complex constraints or a great number of inconsistencies. With these techniques, we can certainly get back consistent answers to our queries. The success of aggregate queries is less certain, but the chances are still reasonable.

Some techniques report uncertainties, and others ignore anything about which we are uncertain. Ultimately, how to handle the situations comes down to the decision of the database administrator or designer. He or she will need to choose the solution he or she feels appropriate to the situation.

References

[AFM06] P. Andritsos, A. Fuxman, and R. Miller. Clean answers over dirty databases: A probabilistic approach. *Proceedings of the 22nd International Conference on Data Engineering*, 2006.

[BC99] L. Bertossi and J. Chomicki. Consistent query answers in inconsistent databases. *Proceedings of the eighteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, 1999.

[M. 99] M. Arenas, L. Bertossi, and J. Chomicki. Consistent Query Answers in Inconsistent Databases. *Proceedings of the eighteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 68–79, 1999.