Lecture 32 — Recovery: Repair, Probability

Jeff Zarnett
jzarnett@uwaterloo.ca

Department of Electrical and Computer Engineering
University of Waterloo

September 9, 2023

Probabilistically Answering Queries

Residues and repairs are not the only way to return consistent answers.

When there are several options, 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.

Both candidate databases will receive a probability of being the “correct" database.

Remember Salaries?

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

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.

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

Are We Sure?

Suppose the query being asked were all the names of all employees making more than $70 000.

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.

Query Modification

Or: 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 rebuilt query reads:

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

The Possibilities Multiply

In a more complex query, we simply multiply the probabilities.

```
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)`.

The group statement is `GROUP BY o.id, c.id`.

Determining Probabilities

The most pressing question is how we determine the probabilities.

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.

Example: Inconsistent Customers

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

The most common values in the database are probably the correct ones.

Inconsistent Data Probability

The algorithm shown above 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.

For data for which there is no obvious distance measure, we term them categorical data; information loss as the distance metric.

Information loss is simply a measure of the difference between the tuple in question and the representative.

Probabilistically Imperfect

The strategy presented is imperfect, however, since we might fail to produce clean answers for some classes of query.

```
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.

Choosing the Representative Data?

Open question: how we might find a representative sample when it is not obviously in a majority-rules scenario.

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.

Popular Answer...

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 previously.

Computational complexity may disqualify a method from being practically useful, should it take too much time to reach a reasonable answer.

Complexity of Repairs

Assume 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.

Should any of these constraints not hold, the problem is pushed into the realm of co-NP-complete problems.

Complexity of Query Transformation

The process of query transformation is shown to have a polynomial time computability of result tuples.

The transformed query will be first order as long as the original query is as well.

The query transformation does not require examining all the possible repairs.

We can evaluate a query with an exponential number of possible repairs in polynomial time.

Aggregate Query Transformation

Aggregate queries: build a conflict graph; a standard graph with nodes and edges.

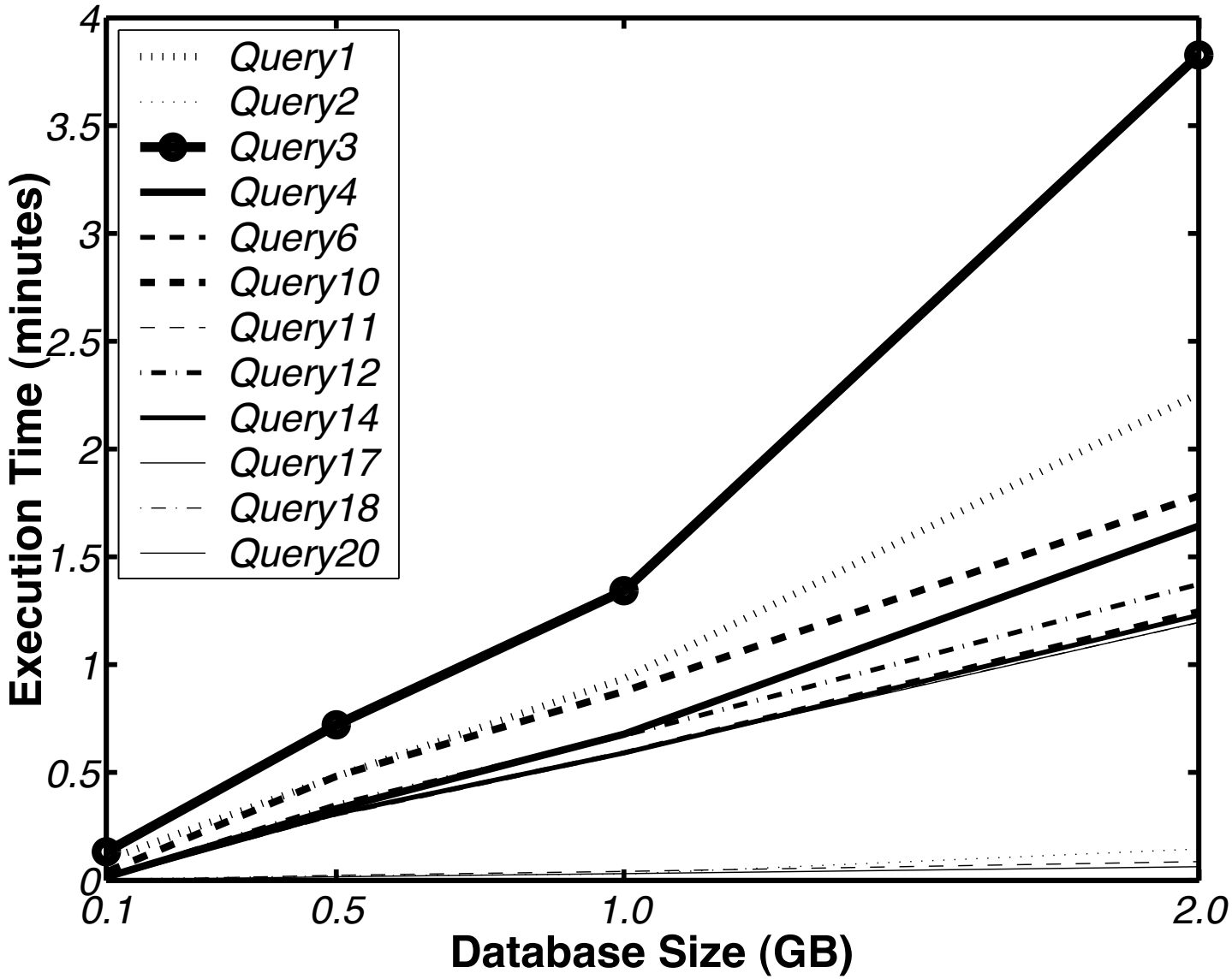
In the graph, maximal independent sets - that is, sets that are the farthest apart in terms of data equality - represent possible repairs of the database.

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.

Probabilistic Complexity



Future Work

To get completeness for disjunctive and existential queries, they will need to move beyond simple query transformations.

Database repairs examined are based on differences against the whole tuple and does not permit modifying attributes of tuples.

The notes cover various deficiencies in the papers that went into this topic...

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.

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.