Query Perturbation Analysis: An Adventure of Database Researchers in Fact-Checking

Jun Yang
Duke University
junyang@cs.duke.edu

Pankaj K. Agarwal
Duke University
pankaj@cs.duke.edu

Sudeepa Roy
Duke University
sudeepa@cs.duke.edu

Brett Walenz
Duke University
bwalenz@cs.duke.edu

You Wu
Google Inc.
wuyou@google.com

Cong Yu
Google Inc.
congyu@google.com

Chengkai Li
University of Texas at Arlington
cli@cse.uta.edu

1 Introduction

Consider a database query with a number of parameters whose values can be specified by users. *Query perturbation analysis* examines how the result of this query varies in response to changes in its parameter values (while the database stays the same). Query perturbation analysis allows us to evaluate the sensitivity of a query's result to its parameters. For example, if we use queries over data to inform decisions or debates, we naturally would like to know whether the conclusions are "brittle" with respect to particular parameter choices. Moreover, by constructing a broader context formed by the results of a query over a large parameter space, query perturbation analysis allows us to further explore this context, such as searching for parameter settings that lead to robust decisions or convincing arguments.

An especially compelling application of query perturbation analysis is in *computational journalism* [10, 9]—specifically, how to fact-check claims derived from structured data automatically. Indeed, this application was what initially drew the authors of this paper to the problem of query perturbation analysis (and to coin this term in [40, 41]), as part of a long-term collaboration among a diverse group of researchers and practitioners from computer science, journalism, and public policy, across Duke University, University of Texas at Arlington, Stanford University, and Google. This paper provides an overview of our research on query perturbation analysis and its applications to journalism, and describes our collective experience and lessons learned—largely from the angle of database researchers.

The connection between fact-checking and perturbation analysis may not be immediately obvious. To start, we observe that in this age of data ubiquity, our media are saturated with claims of "facts" made from data, by politicians, pundits, corporations, and special interest groups alike. Based on data and often quantitative in nature, these claims have seemingly stronger credence in the eyes of the public. Indeed, we can verify their correctness by posing corresponding queries over the underlying data and ensuring that "the numbers check out." However, fact-checking goes beyond verifying correctness. Many claims can be correct yet still mislead. As an old saying goes, "if you torture the data long enough, it will confess to anything" [20]. Think of a claim—e.g., "adoptions went up 65 to 70 percent" when Giuliani "was the New York City mayor"[1]— as a query over data.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

[1] www.factcheck.org/2007/05/levitating-numbers/

The data (here, historical adoption numbers) may be pristine and the form of the query (comparing adoption numbers from two time periods) is completely innocuous. However, the choice of query parameters (comparing 1996–2001 and 1990–1995, which was not made explicit by the original claim) still controls what aspect or subset of the data to present in the claim, and can heavily influence the query result and hence the conclusion of the claim. Query perturbation analysis helps detect and counter such "cherry-picking" practices, because it puts the original claim into a larger context of perturbations for scrutiny; e.g., what if we compare the numbers from the mayor's first and second terms? In Section 2, we show how to apply perturbation analysis to this and other fact-checking tasks, as well as the complementary problem of finding leads from data while avoiding pitfalls.

Query perturbation analysis is perfectly suited to automation. Naively, we can evaluate the query with each possible parameter setting independently, so computation is embarrassingly parallel. In practice, however, brute-force solutions may not be acceptable because the number of possible query parameter settings can be daunting even if the database itself is small in size. In Section 3, we give an overview of techniques for efficient and scalable perturbation analysis. While there has been little prior work from the database community that directly addresses perturbation analysis, we note that some well-studied problems can be framed as perturbation analysis for specific forms of queries. On the other hand, even if we already understand very well how to process a query itself, its perturbation analysis often involves new technical challenges. There is a trade-off between the generality and efficiency of techniques for specific query forms: while we can usually do better with algorithms highly specialized for the occasion, there is a strong motivation to develop techniques and systems that can benefit any database queries. How to close the efficiency gap between general and specialized techniques remains an open challenge.

Interestingly, over the course of our research, we have discovered opportunities to transfer techniques for perturbation analysis to traditional database query processing (and back). Conceptually, we can write perturbation analysis of a query template q as a complex database query, starting with a subquery that enumerates all possible parameter settings for q, then "joining" it with a subquery that computes q for a given parameter setting (with parameters serving as join attributes), and finally applying some aggregation to analyze all results over the entire parameter space. Specifically, Section 3.3 describes some results we obtained exploring this connection, which include new optimizations for OLAP queries inspired by perturbation analysis. This pleasant surprise is a reminder of how pursuit of novel applications could lead to new advances for classic problems.

Last but not least, we reflect on our experience working on perturbation analysis and its applications to journalism in Section 4. We give a brief history of our collaboration with journalists, highlight a few interesting applications we built, and discuss the broader challenges we faced when dealing with the complexity of reality. We then conclude with an outlook for the future research on query perturbation analysis.

2 Modeling Fact-Checking and Lead-Finding

We use a couple of real-life examples to illustrate how to apply query perturbation analysis to a variety of fact-checking and lead-finding tasks. We shall focus on modeling these tasks conceptually for now, and leave the discussion of how to solve them efficiently to Section 3. Our first example is from the domain of US politics.

Example 1 (high correlation claims about voting records): *Evan Bayh, a Democrat from Indiana, ran for the US Senate in 2016. An ad aired during the 2016 elections attacked his records while serving in the Senate earlier, for voting with President Obama 96% of the time.*[2] *Here, the apparently high agreement percentage was used to invoke the impression that Bayh was hyperpartisan and a blind supporter of Obama.*

Ignoring a number of details for now, let us assume that we already have in our database the history of how each senator voted with Obama over time. Then, we can model the above claim as a SQL aggregation query about Bayh's history in particular. The query is parameterized by the time interval of comparison, although the

[2] `www.youtube.com/watch?v=vXEM64l9yfM`

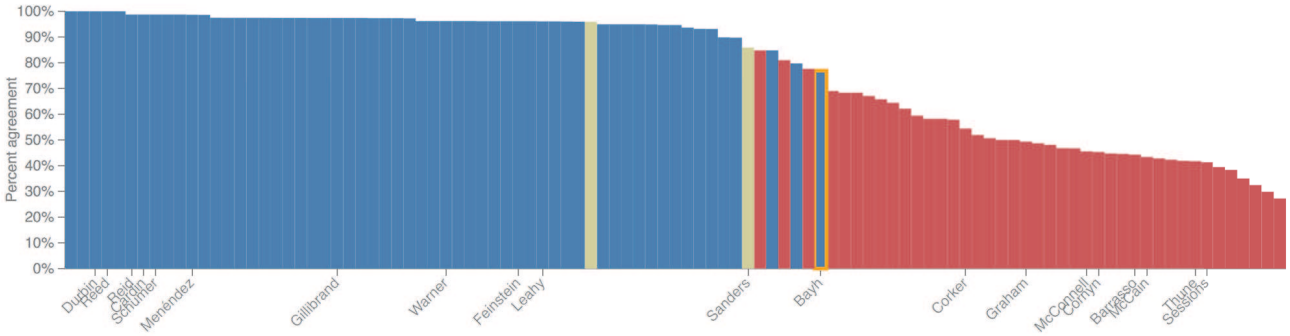

Figure 1: Visualization of how US senators voted with President Obama in 2009, generated by `congress.uclaimicheck.org`. Blue bars represent Democrats; red bars represent Republicans; two brass bars represent independents. The bar representing Evan Bayh is additionally outlined in orange. Data on votes in the Congress were from `GovTrack.us`, and data on the President's positions on these votes were from the *Congressional Quarterly*.

claim does not specify it. Such ambiguity is prevalent in political claims, and is often the very first challenge that fact-checkers must address, oftentimes with little to no help from those who made the claims. With perturbation analysis, however, we can "reverse-engineer" the query by searching for parameter settings that result in the claim's conclusion. In this example, we would look for plausible time intervals over which the aggregation query happens to return 96% (or something close).

As it turns out, the missing time interval in this query is the year 2010, the last year when Evan served in the Senate. The number 96% "checks out." However, as motivated in Section 1, a correct claim can still mislead. First, if we perturb the time interval slightly, to the year before, Bayh would have agreed with Obama only 77% of the time. Considering 2009–2010 would have resulted in 85%, still nowhere as impressive as 96%. In other words, perturbation analysis can reveal when a claim is not robust*, i.e., even small parameter changes can weaken its conclusion significantly.*

We can additionally or alternatively parameterize the same query by the senator we are comparing with Obama. Perturbation in this dimension allows us to see where Bayh stands in a larger context of all senators. Figure 1 (a screenshot from a website we built for the 2016 US elections) visualizes the result of this analysis. We see that in 2009, Bayh was in fact the Democrat who agreed least with Obama, and even less than three Republicans. For a Democrat, 77% agreement with Obama was really low, and 96% was not that unusual. In other words, perturbation analysis can reveal when a claim is not unique*, i.e., it is easy to make similar claims for many other entities besides the one being highlighted.* ■

Claims like the one above have been used repeatedly in political campaigns over the years, by both sides of the political aisle, for both support and attack. For instance, in [40], we used an example from the 2010 elections where an ad supporting Jim Marshall, a Democratic incumbent from Georgia, claimed that he "voted the same as Republican leaders 65 percent of the time." This ad was actually in response to an earlier ad attacking Marshall for voting with Nancy Pelosi (a notable Democrat who was the Speaker of the House at that time) "almost 90 percent of the time."

We should note that, by our observation, human fact-checkers do intuitively perform perturbation analysis, albeit manually (and perhaps instinctively rather than explicitly)—they often put a given claim in a larger context to probe its weaknesses and look for counterarguments. In this sense, perturbation analysis is a natural formalization of human intuition into a computational framework. This formalization brings a number of important new benefits, however. First, we can now elucidate and quantify various intuitive measures of claim quality, such as *robustness* and *uniqueness* motivated in Example 1. Second, we can formulate a variety of fact-checking tasks as computational problems. We have already mentioned in Example 1 the task of *reverse-engineering* claims to remove ambiguity. Another example is the task of *finding counterarguments* to claims. Intuitively, for a claim that is not robust, effective counterarguments would be small parameter changes that significantly weaken the

original claim's conclusion; for a claim that is not unique, an effective counterargument would be a collection of perturbations with similar or stronger conclusions in order to show that the original claim is not special. These tasks can be formulated as search or summarization problems in the space of perturbations. For more details about our framework, see [40, 41].

It is also natural to turn the problem of fact-checking around, and ask how we can find high-quality "leads" from data—claims that are not only interesting, but also robust and unique so they are not easily countered. To illustrate, we use an example about a lighter subject, sports journalism.[3]

Example 2 (one-of-the-few claims in sports): *Sports reporting and commentaries are often inundated with claims involving statistics that supposedly show how impressive a player or performance is. (For a good laugh, listen to the late Frank Deford's opinion on the NPR's Morning Edition.*[4]*) Consider, for example, the claim that "only 10 players in the NBA history had more points, more rebounds, and more assists per game than Sam Lacey in their career" [43]. Sam Lacey was indeed a great basketball player, but most fans probably would not rank him among the most legendary players as the impressive-sounding "only 10" part of the claim would suggest.*

Using perturbation analysis, we can model this claim as a counting query, parameterized by the player, which tallies the number of players who dominate the given players on all three attributes: points, rebounds, and assists per game. By perturbing the player of interest and checking whether the resulting count is 10 or less, we get to see for how many other players we can make the same or even stronger claims. For this example, it turns out that we do so for more than a hundred players, so the original claim about Sam Lacey is not unique.

Thus, to find unique claims, we would consider all possible parameter settings (players) as candidates. Given a candidate player o, suppose the count of players dominating o is k. To decide whether o is unique, we would count τ, the number of parameter settings that result in the same or stronger claims, i.e., the number of players dominated by k or fewer other players. If τ is small, then o is unique. In other words, instead of testing k, which is an unreliable indicator of a claim's interestingness, we test τ, which measures uniqueness under the perturbation analysis framework.

Interestingly, the uniqueness measure naturally penalizes claims of this form for comparing too many attributes. As dimensionality grows, dominance in all attributes becomes increasingly rare among players, so it becomes easier to make claims with impressively low dominance count k. But at the same time, τ would grow, so checking τ naturally protects us from making misleading claims that are not unique. ■

We have done a more in-depth study of how to mine these so-called "one-of-the-few" claims from data in [39]. This earlier work was not presented in the perturbation analysis framework, but the connection, especially to the uniqueness measure of claim quality, should be obvious. This work also allowed claims to compare different subsets of attributes. Although SQL queries traditionally would not consider the attributes they reference as "parameters," we can view changes to the referenced attributes as a general form of query perturbation, and our framework can be extended to accommodate perturbations in not only parameter values but also query forms.

So far, our description of query perturbation analysis has been mostly informal. Before proceeding to the discussion on computational techniques in the next section, we first lay out some formal definitions.

Definition 1 (Query Perturbation Analysis [37]): *Consider a (fixed) database instance $\mathcal{D}$, and a parameterized* query template *q over $\mathcal{D}$ with parameter settings drawn from a* parameter space *P. Let $q(p)$ denote the result of evaluating q with parameter setting $p \in P$ (over $\mathcal{D}$; omitted for brevity). Let $\mathcal{R} = \{(p, q(p)) \mid p \in P\}$ denote the collection of results (*perturbations*) obtained by evaluating q for all possible parameter settings. A*

[3] For an even lighter subject (though less related to journalism), we demonstrated at *SIGMOD* 2014 a system called *iCheck* that made interesting claims about the publication records of database researchers [43] (it also covered other less frivolous domains, of course). For example, *iCheck* found that one author of this paper had a great year of publications in *SIGMOD*, *VLDB*, and *ICDE*—only 8 other researchers had ever managed to publish as much as (or more than) him in one year in every one of these three venues. Then, *iCheck* turned around and noted that the same claim could be made for 89 other researchers (i.e., each of them also had a great year with publication records matched or dominated by no more than 8 others)!

[4] `www.npr.org/templates/story/story.php?storyId=98016313`

query perturbation analysis problem *is specified by a* 4*-tuple* $(\mathcal{D}, P, q, \chi)$*, where* χ *is a* post-processing function *that computes from* $\mathcal{R}$ *the final* answer set *for the analysis.* ■

As an example, we formalize the fact-checking task of *evaluating claim robustness* as follows. Let p_0 denote the parameter setting used by the claim we would like to check. To spell out χ, we need two helper functions:

- The *parameter sensibility function* $\mathsf{SP}(p, p_0)$, where $p \in P$, measures how "sensible" a perturbed parameter setting is with respect to the original. Not all perturbations are equally useful to investigation: we would only assign non-zero sensibility to perturbations that are relevant to the context of the claim, and we assign higher sensibilities to those that are more "natural" (e.g., using "2 years" in a statement is more natural than "22 months").

- The *result strength function* $\mathsf{SR}(q(p), q(p_0))$ measures how much the result of the perturbation deviates from the original. A negative $\mathsf{SR}(q(p), q(p_0))$ means the perturbation weakens the original claim.

Then, we can assess the robustness of the claim using $\chi(\mathcal{R}) = \sum_{(p,q(p))\in\mathcal{R}} \mathsf{SP}(p, p_0) \cdot (\min\{0, \mathsf{SR}(q(p), q(p_0))\})^2$. One way of interpreting this definition is to picture yourself as a "random fact-checker," who randomly perturbs the parameter setting according to sensibility—higher $\mathsf{SP}(p, p_0)$ means p is more likely to be chosen. Here, $\chi(\mathcal{R})$ computes the mean squared deviation from the original claim (considering only perturbations that weaken it) as observed by the random fact-checker—a high $\chi(\mathcal{R})$ means the claim is not robust.

As another example, consider a lead-finding task: *mining one-of-the-few claims with high uniqueness* (recall Example 2). We can generally define the uniqueness of a claim with parameter setting p_0 by computing $\frac{1}{|P|}\sum_{p\in P} \mathbf{1}(\mathsf{SR}(q(p), q(p_0)) < 0)$, i.e., the fraction of parameter settings for which the claim would be weaker than that for p_0. In this case, P is the set of objects of interest, $q(p)$ computes the number of objects dominating $p \in P$ on the given set of attributes for comparison, and $\mathsf{SR}(q(p), q(p_0)) = q(p_0) - q(p)$. Hence, to find claims with uniqueness at least $1 - \frac{\tau}{|P|}$, we use $\chi(\mathcal{R}) = \{(p, q(p)) \mid \tau \geq \sum_{(p',q(p'))\in\mathcal{R}} \mathbf{1}(q(p') \leq q(p))\}$.

We refer interested readers to [41] for more details as well as examples of formulating other fact-checking and lead-finding tasks in this framework. Besides the problems above, one notable practical issue, when finding counterarguments and mining leads, is that there may be too many candidate perturbations to return. One could simply return those that score the highest by some quality measure, but they may be too similar to each other, and some may be outliers and not representative of their neighboring perturbations. Therefore, in [42], we studied the problem of how to choose a small, diverse set of high-quality representatives from the space of perturbations. In general, much research is still needed on effectively analyzing and summarizing the context provided by the space of perturbations, which can require more complex post-processing functions than database queries.

3 Computational Techniques for Perturbation Analysis

A naive way of performing perturbation analysis is to first generate $\mathcal{R}$ by computing $q(p)$ for each possible parameter setting $p \in P$, and then evaluate $\chi(\mathcal{R})$. As mentioned in Section 1, this method is not feasible if P is large, especially when it involves multiple dimensions. In this section, we overview some techniques for more efficient perturbation analysis.

3.1 Query-Specific Methods

Interestingly, a number of well-studied problems in databases can be cast as specific query perturbation analysis tasks, so for these tasks, we can apply known techniques. For example, consider *frequent itemset mining* [6]. Let query template q, parameterized by an itemset T, count the number of transactions that contain all items in T. Then, finding all frequent itemsets is equivalent to the perturbation analysis task of selecting all parameter settings T for which $q(T)$ is above a threshold. For another example, recall Example 2 but consider instead

the simpler lead-finding task of identifying all players who are dominated by no more than k players. If we represent each player as a point in a space whose dimensions are the attributes being compared, then this task essentially amounts to computing the *k-skyband* [27] (a concept that generalizes *skyline* [7]) for a set of points in a high-dimensional space.

Much of perturbation analysis is not covered by existing work, however. For example, to find truly robust and unique claims, we often introduce criteria in the post-processing function χ that require different algorithms or adaptations of techniques from existing work. For instance, as motivated in Example 2, claims about players with low dominance counts are not necessarily interesting; rather, we should look for unique claims—ones that cannot be made for too many players. Hence, the problem shifts from that of computing the k-skyband for a given k, to that of computing the k-skyband for the maximum k such that the k-skyband still contains no more than τ points. Techniques for this new problem setting had to be developed [39].

We observe that practical applications of perturbation analysis often add some element of optimization in the post-processing function χ. Even if efficient evaluation of query template q has been thoroughly studied, the combination of q and an "optimizing" χ leads to interesting new problems, as the following example illustrates.

Example 3 (vote correlation claims and convex hulls; adapted from [41]): *Recall Example 1. Suppose we want to refute the claim about Bayh agreeing too much with Obama with a counterargument showing that the claim is not robust. We can search for this counterargument with an "optimizing" χ. Specifically, consider perturbing the time interval being compared by the claim. We would like to find the interval that minimizes the percentage agreement, subject to some constraints such as length and scope (we omit the details here).*

Putting aside χ for the moment, the query template q in this case is simply range aggregation. The standard technique of prefix sums *allows us to answer q for any query interval in constant time: by precomputing and remembering the running count of agreements over time, we can find the number of agreements over any interval by taking the difference between the values of the running count at the two interval endpoints. As Figure 2 shows, if we plot the running count over time, the percentage agreement over an interval is simply the slope of the line connecting the two endpoints.*

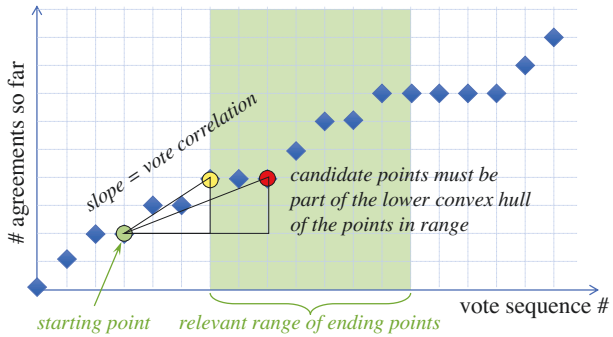

Figure 2: Minimizing percentage agreement by considering the convex hull of points formed by the running count of agreements over time.

However, solving the constrained optimization problem efficiently requires some new insights. Instead of considering all intervals, our technique in [41] aims at quickly finding the optimal interval among those with the same starting point (without explicitly examining all such intervals), and then choosing the best among all possible starting points. Given a starting point, we can determine the relevant range of ending points (as dictated by any given constraints). The key to speeding up the search for the optimal ending point in this range is a geometric intuition illustrated in Figure 2: any candidate must be part of the lower convex hull of the points in the range; there is no need to search the other points. Thus, the problem reduces to indexing for range convex hull queries (see [41] for details). ■

There are many other examples where considering perturbation analysis for well-studied query types breathes new life into old problems. While it is both interesting and useful to develop highly specialized solutions for specific query templates, they require a lot of human expertise and effort. We may not have such a luxury when so many queries and application domains could benefit from perturbation analysis. Thus, it is also important to devise more generic methods—which we discuss next.

3.2 Generic Methods

Generic methods aimed at speeding up the computation of $\mathcal{R}$ include *parallelization* (i.e., evaluating all perturbations in parallel); *grouping* (i.e., evaluating some perturbations together efficiently as a group); and *memoization* (i.e., caching the result of a piece of computation and reusing it later). The classic database problem of *MQO* (*multiple-query optimization* [29]) is closely related, although it traditionally targets very different query workloads. Perturbation analysis typically involves far more queries than MQO, but all of them have the same form and differ only in their parameter settings; MQO makes no such assumption and hence focuses more on finding common subqueries to share processing. Another key difference is that perturbation analysis has a post-processing function χ, which opens up the possibility of "pushing down" selection criteria from χ down into the computation of $\mathcal{R}$. These additional selection criteria in turn enable more *pruning* (i.e., using cached results for one set of inputs to help discard unpromising parameter settings without full evaluation). We illustrate the use of memoization and pruning with a simple example below.

Example 4 (memoization and pruning for finding one-of-the-few claims; adapted from [37]): *Recall Example 2. Suppose our task now is to find all claims of the form "player A's performance in stats X and Y during season S is dominated no more than* 10 *times." A baseline approach would be to process one player-season pair at a time, looking up the performance record (say (x, y)), counting the number of records dominating (x, y), and then checking whether this count is no more than* 10*. With memoization, we remember the outcome of this check together with the input—importantly, (x, y), instead of the player-season pair. Later, if we encounter another player-season pair that happens to yield the same (x, y), we can reuse the result of the check. How often does this simple memoization method help? The answer depends on how many distinct (x, y) pairs there are versus player-season pairs. If the number of players is large while stats have relatively smaller active domains, memoization can produce great savings—more than 90% reduction in the number of counting queries for some practical scenarios tested in [37].*

We can further improve the above approach with pruning. Once we find that a performance record (x, y) has a dominance count of c, we remember (x, y, c). Upon encountering a previously unseen record (x', y'), instead of immediately computing its dominance count, we first check the entries we remembered to see if we find some entry $(x^\star, y^\star, c^\star)$ where $(x^\star, y^\star)$ dominates (x', y') and $c^\star \geq k$. If yes, then we know, by transitivity of dominance, that (x', y') must have a dominance count greater than k and therefore can be discarded without further processing. ■

Note that even after applying memoization and pruning, as in the example above, the resulting procedure still may not be as efficient as a specialized method, e.g., a handcrafted k-skyband algorithm. However, the advantages of generic methods are their broader applicability and the possibility for automatic optimization, which significantly reduces development costs. How to close the efficiency gap between the general and specialized solutions is an ongoing challenge, which requires continuing to develop not only new generic methods but also better automatic optimization—a topic that we will come back to in Section 3.4.

One issue that we glossed over in Example 4 is how to come up with the pruning logic in the first place. If we must specify this logic by hand for every perturbation analysis task, some appeal of the generality of pruning would be lost. Fortunately, as we shall see in Section 3.3, sometimes we can derive the pruning logic automatically from a specification of the perturbation analysis task (e.g., for Example 4). However, for more complex pruning opportunities (see [37] for examples), identifying them automatically remains an open challenge.

Another suite of generic methods for perturbation analysis aim at exploiting properties of the post-processing function χ that commonly arise in applications. First, for many fact-checking and lead-finding tasks, χ uses a parameter sensibility function SP (introduced at the end of Section 2), which intuitively weighs some perturbations more than others. Instead of considering perturbations in some arbitrary order, a good heuristic would be to consider them in the decreasing order of sensibility, and stop when we are sure that remaining perturbations will no longer contribute to the answer because of low sensibility. Second, consider the case where χ involves

searching for the "best" parameter settings within a region of the parameter space defined by constraints. While optimization over the entire region may seem complex, sometimes we can divide the region into "zones" within which we know how to solve the optimization problem more efficiently (e.g., as in Example 3). In [40, 41], we developed various "meta" algorithms implementing these high-level generic methods, with pluggable low-level building blocks that can be customized for the occasion.

An exciting direction that we have been pursuing recently is the use of approximation methods for perturbation analysis. Approximation is effective for taming computational complexity when aggregating or optimizing over a huge, high-dimensional parameter space, and in most practical scenarios, we can tolerate some errors. The most straightforward method would be uniform random sampling, but alternative methods often work better in our context: e.g., *importance sampling* [32] based on the parameter sensibility function SP, and *coreset sampling* [4] for certain types of post-processing functions. We also borrow a number of ideas from machine learning. Acquiring each sample involves a potentially expensive query, but we can learn a model predicting the contribution of a parameter setting to the result of the analysis. This learned model can be used to inform sampling design, and techniques from *active learning* [30] and *quantification learning* [15] are also applicable. We hope to make our results on this front available soon.

3.3 Connections to Traditional Query Processing

As mentioned in Section 1, we can think of the perturbation analysis of a query template q as a complex query, with one part enumerating all possible parameter settings and the rest computing q for each parameter setting. If the post-processing function χ is not too complicated, it may be expressed as additional aggregation and/or selection. For instance, the two examples at the beginning of Section 3.1, mining frequent itemsets and finding k-skyband, can be written in SQL as follows (for simplicity, consider only 2-itemsets and 2-d skyband):

```
-- Basket(bid, item) stores one item per row
SELECT i1.item, i2.item
FROM Basket i1, Basket i2
WHERE i1.bid = i2.bid
GROUP BY i1.item, i2.item
HAVING COUNT(*) >= s;
```

```
-- Object(id, x, y) stores one object per row
SELECT L.id, COUNT(*)
FROM Object L, Object R
WHERE L.x<=R.x AND L.y<=R.y AND (L.x<R.x OR L.y<R.y)
GROUP BY L.id
HAVING COUNT(*) <= k;
```

These queries are both joins followed by grouping and further selection based on group sizes, which are examples of so-called *iceberg queries* [13] in OLAP. There has been plenty of work on computing frequent itemsets, skybands, and iceberg queries. Interestingly, their techniques have been developed largely disjointly: the techniques for the first two problems are quite specific, while work on iceberg queries does not typically address the joins that happen before grouping and aggregation. Perturbation analysis offers us an opportunity to reexamine these traditional query processing problems in a single framework.

A natural idea to try is applying the generic methods for perturbation analysis to iceberg query processing, which we did in [36]. To apply memoization and pruning, we devise a physical join operator called *NLJP*, or *nested-loop join with pruning*. NLJP operates as a nested loop, evaluating its inner(-loop) query with the join attribute values of each tuple produced by its outer(-loop) query. NLJP caches the results of the inner query by its input join attribute values, and for each new tuple from the outer query, evaluates a pruning predicate to determine whether it can safely skip the inner query evaluation. The cache also enables memoization.

Another style of pruning, which generalizes the *A-priori* technique for frequent itemset mining, works by first applying the HAVING constraint to an input table before it is joined. For example, for the query computing frequent 2-itemset above, we can replace each instance of Basket by

```
(SELECT * FROM Basket WHERE item IN
 (SELECT item FROM Basket GROUP BY item HAVING COUNT(*) >= s))
```

and drastically reduce work in subsequent processing. This optimization can be achieved by query rewriting.

The key technical challenge we tackled in [36] is how to identify and apply memoization and pruning techniques automatically, without relying on any manual hints. To this end, we have developed formal conditions for the applicability of these techniques. Interestingly, both NLJP- and A-priori-style pruning are underlined by query *monotonicity* properties, where we can infer useful relationships between query results when input tables have containment relationships. We analyze SQL queries with the knowledge of the database constraints to detect optimization opportunities. To apply them, we rewrite the query and/or insert NLJP operators into the query plan. Notably, we can deduce pruning predicates used by NLJP automatically, for query conditions involving linear equalities and inequalities, using quantifier and variable elimination methods [21].

Our implementation in PostgreSQL [1] shows substantial performance improvements over existing database systems for complex iceberg queries [36]—a nice example of how insights from perturbation analysis help with classic query processing. Beyond [36], we are actively applying the approximation methods discussed in Section 3.2 to query processing, to reduce the cost of evaluating expensive subqueries or user-defined predicates.

What makes the connection between perturbation analysis and traditional query processing doubly exciting is that it has also led to new breakthroughs for perturbation analysis. With the techniques developed in [36], we can automatically identify and apply memoization and pruning techniques for a fairly broad class of perturbation analysis tasks expressible as SQL iceberg queries. Another interesting follow-up problem is how to apply A-priori-style pruning techniques to perturbation analysis, especially when we generalize perturbations to not only parameter values but also query forms—this pruning style is particularly powerful across changes to the "dimensions" involved in the query.

3.4 System Support

We have built a system called *Perada* [37] to support perturbation analysis of SQL queries. We want this system to be general (supporting any query template expressed in SQL), scalable (capable of processing a large number of perturbations for time-sensitive fact-checking and lead-finding tasks), and easy to use (allowing developers to code new analyses quickly, without burdening them with low-level implementation and tuning). As mentioned earlier, a key challenge we face is the trade-off between generality and efficiency. The approach we have taken with Perada is to support a suite of generic methods including parallelization, grouping, memoization, and pruning, as discussed in Section 3.2, but rely on developers to specify how to apply these methods to the task at hand through a flexible, high-level API. Perada takes care of parallelization and hides the complexity of concurrency and failures from developers.

Another important feature of Perada is automatic tuning. While Perada relies on developers to specify parallelization, memoization, and pruning opportunities, it automatically makes various tuning decisions critical to performance: e.g., whether the benefit of memoization and pruning outweighs their overhead, or how to strike the balance between parallelism and serialism (the latter enables memoization and pruning). These decisions are difficult for developers to make, as the various methods interact with each other in subtle ways, and their effectiveness generally depends on factors such as data characteristics. We use an example to illustrate this point.

Example 5 (factors influencing pruning effectiveness; adapted from [37]): *Consider the pruning method in Example 4 for finding one-of-the-few claims. We apply it to a dataset about the Major League Baseball that records player performances in each season in terms of more than twenty stats. First, suppose we are interested in three stats: hits, home runs, and stolen bases, and the dominance count threshold for reporting a claim is* $k \leq 50$*. Figure 3 (ignore the dashed line plot for now) shows the pruning rate over the course of examining all player-season pairs sequentially in random order. We see that the pruning rate picks up rapidly at the beginning of execution, and converges to a very high percentage. Intuitively, as we see more perturbations, we build up a better picture of the "decision boundary" separating perturbations inside and outside the answer set, eventually allowing us to prune most perturbations lying on one side of the decision boundary.*

Now consider the same analysis on the same dataset with the same pruning method, but with $k = 5000$ and another three stats of interest: hits allowed, strikeouts, and earned run average. The pruning rate over time for this case is plotted (with dashes) in Figure 3 for comparison. We see the pruning rate here picks up more slowly, and converges to a lower percentage. Not only is k bigger in this case, but it also turns out that strikeouts and earned run average are anti-correlated. Together, these factors lead to a more complex decision boundary that requires more samples to acquire, as well as a much lower percentage of perturbations that can be pruned.

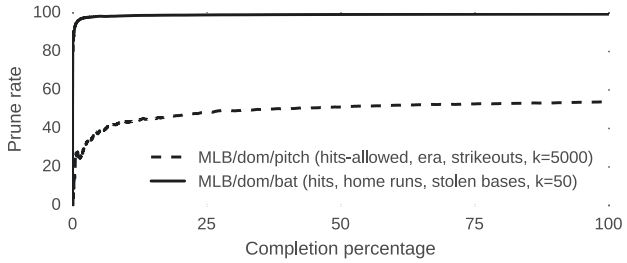

Figure 3: (Adapted from [37].) Pruning rate over the course of serially executing the analysis in Example 5, for different thresholds of k and different stats of interest.

This difference in the pruning rates suggests that the parallel implementations of the two cases above need to be tuned differently to achieve their best performance. When we parallelize the analysis using multiple workers each responsible for a subset of the parameter space, workers can exchange locally cached information to help improve each other's pruning rate. However, some exchanges incur overhead, and their benefits to pruning generally diminish over time, and at different speeds for different workloads. Therefore, workers should exchange information more often early during the execution, and less as time progresses. Furthermore, workers in the first case above can stop exchanging information earlier than in the second case. ■

Thus, when applying parallelization, memoization, and pruning, Perada automatically tunes their performance through run-time observation, learning, and adaptation. To be more specific, Perada divides the parameter space of the perturbation analysis into units, and processes them in parallel on a cluster of workers (using *Spark* [14]). Perada provides both a global key/value cache (using *Redis* [22]) for memoization, and a SQL cache at each worker (using *SQLite* [2]) for pruning, which typically involves more complex queries against the cache. Information exchange through the global key/value cache is immediate, but the contents of all local SQL caches are synchronized only at the end of each execution "epoch," to keep the overhead low. Behind the scene, Perada dynamically and adaptively controls the timing of epochs as well as what combination of optimizations to apply in each epoch. For additional details about Perada, see [37].

From our experience and experiments with real-life applications of perturbation analysis, Perada significantly simplifies development efforts for ad hoc analysis, and delivers vastly better performance than other general solutions based on existing systems. One quibble is that Perada is still not fully declarative; for example, developers have to spell out the pruning logic. As discussed in Section 3.3, we are able to derive pruning logic automatically for some forms of perturbation analysis tasks, but in general, much research is still needed in automatic optimization of declarative analysis.

4 Reflection and Outlook

This paper presents our recent research on query perturbation analysis. We have focused on its applications in fact-checking and lead-finding, and offered an overview of its computational techniques and its connection to traditional query processing. Looking back at our journey, we realize how incredibly lucky we were—we had the chance to collaborate with visionary colleagues in journalism and public policy to work on the important and timely problem of fact-checking, yet, at the same time, the insights and techniques we developed for the problem contribute back to core computer science. Finding new solutions to well-studied, classic database problems, as discussed in Section 3.3, came as a pleasant surprise. Perhaps even more amazing is the experience that we are able to explain the results in our *SIGMOD*, *VLDB*, and *KDD* papers to non-technical friends at parties or strangers on commute, and have them genuinely interested in this research.

In the following, we give a brief chronology of our journey, discuss the broader challenges we faced when building real-world applications, and offer our outlook on the future research in query perturbation analysis.

Project History and Practical Deployments One author of this paper, Yang, was introduced to computational journalism more than ten years ago by James T. Hamilton, a public policy colleague at Duke at the time (now director of the journalism program at Stanford). However, concrete ideas for connecting computational journalism to database research took time to develop, partly because of the overwhelming number of possibilities—computational journalism is so broad that virtually every field of computing can help in various ways. In 2009, Hamilton co-organized a summer workshop on computational journalism at the Center for Advanced Study in the Behavioral Sciences at Stanford, and invited Yang and Sarah Cohen, a seasoned investigative reporter who joined Duke faculty from the Washington Post. This workshop deserves much credit in developing the field; it also helped Yang and Cohen find a focus on data-driven investigative reporting. In parallel, two other authors of this paper, Li and Yu, had begun working on mining interesting facts from data in domains such as sports and weather. A discussion between Yang and Li in the summer of 2010, when both were visiting HP Labs, brought the group together, and identified the connection between fact-checking and lead-finding, as well as the pitfalls of correct but uninteresting or even misleading claims. The group co-authored a vision paper [10] in *CIDR* 2011, in which one could see initial ideas about perturbation analysis starting to take shape.

The project then took off with a diverse research team. Agarwal, an algorithms researcher specializing in computational geometry, added the geometric insights and algorithmic rigor to perturbation analysis. The first PhD student to work on perturbation analysis, You, was co-advised by Yang and Agarwal and focused primarily on the modeling and algorithmic aspects; after graduation, You continued to work on projects related to fact-checking in Cong's group at Google Research. Walenz will be the second PhD student to graduate with a dissertation on perturbation analysis, focusing on system support, practical applications, and most recently, exploring its connection with traditional database query processing together with Roy, a database researcher specializing in theory. Bill Adair, a journalist and fact-checker who founded PolitiFact, joined the Duke faculty in 2013, and has kept the project solidly grounded in its practical fact-checking applications. In retrospect, we recognize that the luck we had was only possible with patience and persistence, as well as a team of collaborators with diverse and complementary backgrounds but the same commitment to practical impact.

Over the years, we have developed a number of applications of perturbation analysis, and worked with real data, real users, as well as journalists and professional fact-checkers. As mentioned earlier, the *iCheck* system, demonstrated at *SIGMOD* 2014 [43], found and checked claims about player performances in professional baseball, publication records of database researchers, and voting records of the US Congress. *FactWatcher*, demonstrated at *VLDB* 2014 [17], found and monitored claims about professional basketball and weather. For the 2016 US elections, we built a website *congress.uclaimicheck.org*. It analyzed the congressional voting records from January 2009 to September 2016, and let visitors compare how legislators voted with party majorities and the President, and more importantly, explore how the comparison stacked up under different contexts using perturbation analysis—over time, among groups of peers, and for "key votes" identified by lobbying organizations. It was demonstrated at the 2016 *Computational+Journalism Symposium* [35] and released to the public in September 2016. Finally, we have been working with Duke Athletics on a system for finding interesting "factlets" about each Duke men's basketball game. It runs after each game, and uses perturbation analysis to produce a variety of factlets about player performances in that game within minutes. As of May 2017, these factlets have become a part of the official Duke men's basketball stats website, available for fans to explore and share on social media.

Challenges in Computational Fact-Checking It has been a humbling experience for us to see the wide gamut of knowledge, skills, and efforts required of journalists in the practical applications that we have worked on. Computational fact-checking poses a broad spectrum of challenges; query perturbation analysis is but one piece of the puzzle. We noticed that when discussing our work with other computer scientists, the first reaction we got

of the form "are you working on X" referred to a wide range of X's. Natural language processing is one of the most often mentioned. Indeed, we need it to map a natural language claim to a query template q. Nonetheless, as seen in Section 2, perturbation analysis can help disambiguate claims; it would be interesting to further explore the idea of using data contents to help understand natural language claims based on them.

Source discovery, data extraction, data integration, and data cleaning remain very labor-intensive tasks. As an example from our experience, even in the case of high-quality congressional voting records, it was a pain collecting and linking other related data, such as key votes published by lobbying organizations. References are often incomplete or ambiguous—especially when many roll calls are associated with the same bill—and worse, occasionally contain typos. Some of them could only be resolved by human experts. While automated data extraction and cleaning techniques have come a long way, attaining the level of accuracy required for fact-checking specific domains without a lot of data—let alone expert-labeled data—remains a challenge in practice. One specific problem that we have been working on recently is how to focus cleaning efforts on subsets of data that matter the most to given fact-checking tasks [31]. Given the complexity of real-life data, we are also interested in developing methods for annotating data that can alert us to potential misuses or misinterpretation.

All of the above challenges—and fact-checking in general—require deep domain knowledge. Consider again Example 1. Without domain knowledge, it is not even clear what "voting with President Obama" means—because US Presidents do not vote—and where we can find Presidential positions on congressional votes. "Voting with democratic majority" is even trickier: the "non-voting" members of the House (e.g., those from Washington D.C. or Guam) do cast votes, but rules about when they get to vote and when their votes get counted kept changing over time depending on which party controlled the House. The few such votes probably never swayed the democratic majority, but just knowing we might be able to ignore this issue also requires domain knowledge. How to capture and utilize such knowledge computationally remains an open challenge.

We have also come to learn how the presentation of claims and fact-checks is critically important to the audience. To illustrate, we do not need to go to examples where perception is influenced by ideologies and personal worldviews; just consider two neutral factlets mined from Duke men's basketball games: 1) "He was one of the only ten players who had at least x points and at least y rebounds in a game in Duke history." 2) "He was one of the only ten players who had at least y rebounds in a game in Duke history." Logically, (2) is stronger. From (2) we can simply tack on the number of rebounds he made in that game, no matter how low it is, and obtain a valid factlet in the form of (1). Furthermore, it is much easier to make a factlet like (1) than (2), because it is less likely for points in higher dimensions to form dominance relations; in other words, (2) would have higher uniqueness than (1). Yet, despite this rational analysis, we found that, anecdotally, the majority of users thought (1) sounded more impressive. This simple example illustrates how no amount of modeling can replace real-world testing, and hints at the challenges of using fact-checking to correct human misperceptions.

Computational fact-checking raises many more challenges. For example, there are also the problems of how to find claims to check in the first place, which we worked on extensively [16, 18], how to deliver fact-checks to the public in a timely manner, which we are starting to research on [3], as well as how to effectively support the collaboration of fact-checkers, journalists, and citizens alike. Last but not least, there is "fake news." Our collaboration with fact-checkers began long before "fake news" rose to prominence in research circles following the 2016 US elections. While we had been busy fighting misleading (but still factually correct) claims, sadly, downright blatant lies were starting to have a field day. Combating such lies requires a different suite of techniques, ranging from source credibility, media forensics, and social network analysis. While it is impossible to give a comprehensive survey on computational fact-checking here, we hope that this discussion, however cursory, will be useful to those considering to work on computational fact-checking and combating misinformation and disinformation in general.

Open Problems in Perturbation Analysis Query perturbation analysis is still a young research direction in databases. We have mentioned a number of open problems throughout this paper. First, there is a wealth

of query templates worthy of consideration. Just pick your favorite lie that levitates numbers—and there are plenty of them out there in media (and research papers!)—and you have a candidate for applying perturbation analysis. Such investigations can be useful and interesting in their own right, and more importantly, they help us discover new ways of assessing claims and countering them, as well as new computational methods, which in turn help us generalize the query perturbation framework and optimization techniques. Second, much work is still needed in supporting fully declarative perturbation analysis and in closing the gap between general and specific solutions. Ideally, we would like to integrate into a database system a comprehensive suite of processing and automatic optimization techniques that provide a level of support for perturbation analysis comparable to that for traditional SQL queries. Directions we are pursuing currently include generic approximation methods, as well as query analysis and rewrite techniques for exploiting more sophisticated pruning opportunities.

This paper has discussed the connection between perturbation analysis and traditional query processing, but there are also connections to many other active areas of database research. For example, a large body of work on *uncertain data management* [11, 5] considers the effect of data perturbations on query results. Our study of query parameter perturbations can be seen as a conceptual counterpoint—while uncertain databases consider one query over many database instances (possible worlds), we are also interested in many queries (perturbed versions of each other) over one database instance. There has also been much work on *lineage* and *provenance* (surveyed in [8]), and more recently, *causality* [23, 24, 25, 26]. This line of work also studies how data contribute to query results; depending on how contribution is defined, some problems may be analogous or related to the analysis of query parameter perturbations. Recent work on *query by output* [34] and *view synthesis* [28] can be seen as types of perturbation analysis, where we search for queries with desired properties. Also similar in spirit are [33, 19, 38], though their search spaces and goals are different. Finally, *differential privacy* [12] has an element of understanding sensitivity as well—how an individual data perturbation can affect query results in the worst case. It would be very interesting to further explore potentially deep connections among various types of perturbations of data, parameters, and query forms.

Conclusion The biggest takeaway point about query perturbation analysis can be perhaps summarized by a phrase from the title of You's dissertation—it is a change of perspective "from answering questions to questioning answers and raising good questions," and this change not only introduces new database research challenges but also brings new insights into well-studied problems. Perturbation analysis was initially motivated by fact-checking, and we continue to collaborate closely with fact-checkers and journalists to push the boundaries of computational fact-checking. Speaking from our experience, working as an interdisciplinary team to tackle problems of societal importance can be both practically fulfilling and intellectually stimulating.

Acknowledgments *This work was partially supported by NSF grants IIS-1408846, IIS-1408928, IIS-1565699, IIS-1719054, grants from the Knight Foundation, and a Google Faculty Award. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of the funding agencies.*

References

[1] PostgreSQL. `http://www.postgresql.org/`.

[2] SQLite. `http://sqlite.org/`.

[3] B. Adair, C. Li, J. Yang, and C. Yu. Progress toward "the holy grail": The continued quest to automate fact-checking. In *Proc. 2017 Computation+Journalism Symposium*, Evanston, Illinois, USA, Oct. 2017. Informal publication.

[4] P. K. Agarwal, S. Har-Peled, and K. R. Varadarajan. Geometric approximation via coresets. In J. E. Goodman, J. Pach, and E. Welzl, editors, *Combinatorial and Computational Geometry*, pages 1–30. Cambridge University Press, New York, 2005.

[5] C. C. Aggarwal, editor. *Managing and Mining Uncertain Data*. Springer, 2009.

[6] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *Proc. 1994 Intl. Conf. on Very Large Data Bases*, pages 487–499, Santiago de Chile, Chile, Sept. 1994.

[7] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *Proc. 2001 Intl. Conf. on Data Engineering*, pages 421–430, Heidelberg, Germany, Apr. 2001.

[8] J. Cheney, L. Chiticariu, and W. C. Tan. Provenance in databases: Why, how, and where. *Foundations and Trends in Databases*, 1(4):379–474, 2009.

[9] S. Cohen, J. T. Hamilton, and F. Turner. Computational journalism. *Communications of the ACM*, 54(10):66–71, 2011.

[10] S. Cohen, C. Li, J. Yang, and C. Yu. Computational journalism: A call to arms to database researchers. In *Proc. 2011 Conf. on Innovative Data Systems Research*, Asilomar, California, USA, Jan. 2011.

[11] N. N. Dalvi, C. Ré, and D. Suciu. Probabilistic databases: Diamonds in the dirt. *Communications of the ACM*, 52(7):86–94, 2009.

[12] C. Dwork. A firm foundation for private data analysis. *Communications of the ACM*, 54(1):86–95, 2011.

[13] M. Fang, N. Shivakumar, H. Garcia-Molina, R. Motwani, and J. D. Ullman. Computing iceberg queries efficiently. In *Proc. 1998 Intl. Conf. on Very Large Data Bases*, pages 299–310, New York City, New York, USA, Aug. 1998.

[14] T. A. S. Foundation. Apache Spark: Lightning-fast cluster computing. `http://spark.apache.org/`.

[15] P. González, A. C. no, N. V. Chawla, and J. J. D. Coz. A review on quantification learning. *ACM Computing Surveys*, 50(5):74:1–74:40, Sept. 2017.

[16] N. Hassan, F. Arslan, C. Li, and M. Tremayne. Toward automated fact-checking: Detecting check-worthy factual claims by claimbuster. In *Proc. 2017 ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining*, Halifax, Nova Scotia, Canada, Aug. 2017.

[17] N. Hassan, A. Sultana, Y. Wu, G. Zhang, C. Li, J. Yang, and C. Yu. Data in, fact out: Automated monitoring of facts by FactWatcher. *Proc. the VLDB Endowment*, 7(13), 2014.

[18] N. Hassan, G. Zhang, F. Arslan, J. Caraballo, D. Jimenez, S. Gawsane, S. Hasan, M. Joseph, A. Kulkarni, A. K. Nayak, V. Sable, C. Li, and M. Tremayne. ClaimBuster: The first-ever end-to-end fact-checking system. *Proc. the VLDB Endowment*, 10(12), Aug. 2017.

[19] Z. He and E. Lo. Answering why-not questions on top-k queries. In *Proc. 2012 Intl. Conf. on Data Engineering*, pages 750–761, Washington DC, USA, Apr. 2012.

[20] D. Huff. *How to Lie with Statistics*. W. W. Norton & Company, reissue edition, 1993.

[21] L. Khachiyan. *Fourier–Motzkin elimination method*, pages 1074–1077. Encyclopedia of Optimization, Springer US, Boston, MA, 2009.

[22] R. Labs. Redis. `http://redis.io/`.

[23] A. Meliou, W. Gatterbauer, J. Y. Halpern, C. Koch, K. F. Moore, and D. Suciu. Causality in databases. *IEEE Data Engineering Bulletin*, 33(3):59–67, 2010.

[24] A. Meliou, W. Gatterbauer, K. F. Moore, and D. Suciu. The complexity of causality and responsibility for query answers and non-answers. *Proc. the VLDB Endowment*, 4(1):34–45, 2010.

[25] A. Meliou, W. Gatterbauer, K. F. Moore, and D. Suciu. WHY SO? or WHY NO? functional causality for explaining query answers. In *Proc. 2010 Intl. VLDB Workshop on Management of Uncertain Data*, pages 3–17, Singapore, Sept. 2010.

[26] A. Meliou, W. Gatterbauer, and D. Suciu. Bring provenance to its full potential using causal reasoning. In *Proc. 2011 USENIX Workshop on the Theory and Practice of Provenance*, Heraklion, Crete, Greece, June 2011.

[27] D. Papadias, Y. Tao, G. Fu, and B. Seeger. Progressive skyline computation in database systems. *ACM Transactions on Database Systems*, 30(1):41–82, 2005.

[28] A. D. Sarma, A. G. Parameswaran, H. Garcia-Molina, and J. Widom. Synthesizing view definitions from data. In *Proc. 2010 Intl. Conf. on Database Theory*, pages 89–103, Lausanne, Switzerland, Mar. 2010.

[29] T. K. Sellis. Multiple-query optimization. *ACM Transactions on Database Systems*, 13(1):23–52, 1988.

[30] B. Settles. *Active Learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2012.

[31] S. Sintos, P. K. Agarwal, and J. Yang. Data cleaning and fact checking: Minimizing uncertainty versus maximizing surprise. Technical report, Duke University, 2018. `https://users.cs.duke.edu/~ssintos/DataCleanFactCheck.pdf`.

[32] S. T. Tokdar and R. E. Kass. Importance sampling: A review. *Wiley Interdisciplinary Reviews: Computational Statistics*, 2:54–60, 2010.

[33] Q. T. Tran and C.-Y. Chan. How to ConQueR why-not questions. In *Proc. 2010 ACM SIGMOD Intl. Conf. on Management of Data*, pages 15–26, Indianapolis, Indiana, USA, June 2010.

[34] Q. T. Tran, C.-Y. Chan, and S. Parthasarathy. Query by output. In *Proc. 2009 ACM SIGMOD Intl. Conf. on Management of Data*, pages 535–548, Providence, Rhode Island, USA, June 2009.

[35] B. Walenz, J. Gao, E. Sonmez, Y. Tian, Y. Wen, C. Xu, B. Adair, and J. Yang. Fact checking congressional voting claims. In *Proc. 2016 Computation+Journalism Symposium*, Stanford, California, USA, Sept. 2016. Informal publication.

[36] B. Walenz, S. Roy, and J. Yang. Optimizing iceberg queries with complex joins. In *Proc. 2017 ACM SIGMOD Intl. Conf. on Management of Data*, pages 1243–1258, Chicago, Illinois, USA, May 2017.

[37] B. Walenz and J. Yang. Perturbation analysis of database queries. *Proc. the VLDB Endowment*, 9(14):1635–1646, Sept. 2016.

[38] E. Wu and S. Madden. Scorpion: Explaining away outliers in aggregate queries. *Proc. the VLDB Endowment*, 6(8):553–564, June 2013.

[39] Y. Wu, P. K. Agarwal, C. Li, J. Yang, and C. Yu. On "one of the few" objects. In *Proc. 2012 ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining*, pages 1487–1495, Beijing, China, Aug. 2012.

[40] Y. Wu, P. K. Agarwal, C. Li, J. Yang, and C. Yu. Toward computational fact-checking. *Proc. the VLDB Endowment*, 7(7):589–600, 2014.

[41] Y. Wu, P. K. Agarwal, C. Li, J. Yang, and C. Yu. Computational fact checking through query perturbations. *ACM Transactions on Database Systems*, 42(1):4:1–4:41, Mar. 2017.

[42] Y. Wu, J. Gao, P. K. Agarwal, and J. Yang. Finding diverse, high-value representatives on a surface of answers. *Proc. the VLDB Endowment*, 10(7):793–804, Mar. 2017.

[43] Y. Wu, B. Walenz, P. Li, A. Shim, E. Sonmez, P. K. Agarwal, C. Li, J. Yang, and C. Yu. iCheck: Computationally combating "lies, d—ned lies, and statistics". In *Proc. 2014 ACM SIGMOD Intl. Conf. on Management of Data*, Snowbird, Utah, USA, June 2014.