# Toward Computational Fact-Checking [*]

You Wu[†] Pankaj K. Agarwal[†] Chengkai Li[‡] Jun Yang[†] Cong Yu[§]
[†]Duke University, [‡]University of Texas at Arlington, [§]Google Research

## ABSTRACT

Our news are saturated with claims of “facts” made from data. Database research has in the past focused on how to answer queries, but has not devoted much attention to discerning more subtle qualities of the resulting claims, e.g., is a claim “cherry-picking”? This paper proposes a framework that models claims based on structured data as parameterized queries. A key insight is that we can learn a lot about a claim by perturbing its parameters and seeing how its conclusion changes. This framework lets us formulate practical fact-checking tasks—reverse-engineering (often intentionally) vague claims, and countering questionable claims—as computational problems. Along with the modeling framework, we develop an algorithmic framework that enables efficient instantiations of “meta” algorithms by supplying appropriate algorithmic building blocks. We present real-world examples and experiments that demonstrate the power of our model, efficiency of our algorithms, and usefulness of their results.

## 1 Introduction

While a lot of current database research is devoted to the art of *answering* queries, equally critical to our understanding of data is the art of discerning the “quality” of claims and asking queries that lead to high-quality claims. But first, what does “quality” mean? Consider the following.

**Example 1** (**Giuliani’s Adoption Claim** (from `factcheck.org`)). *During a Republican presidential candidates’ debate in 2007, Rudy Giuliani claimed that “adoptions went up 65 to 70 percent” in the New York City “when he was the mayor.” More precisely, the comparison is between the total number of adoptions during 1996–2001 and that during 1990–1995. Giuliani was in office 1994–2001. The claim checks out according to data, but that does not mean we should stop here. Why does the claim compare these two particular six-year periods? As it turns out, the underlying data reveals that while adoption increased steadily before 1998, it began to slow down in 1998, a trend that continued through 2006. Lumping data into the periods of 1990–1995 and 1996–2001 masks this trend. Comparing the adoption rates at the beginning and the end of his tenure would have only yielded an increase of 17 percent.*

**Example 2** (**Vote Correlation Claim** (from `factcheck.org`)). *A TV ad in the 2010 elections claimed that Jim Marshall, a Democratic incumbent from Georgia “voted the same as Republican leaders 65 percent of the time.”*[1] *This comparison was made with Republican Leader John Boehner over the votes in 2010. If we look at the history since 2007, however, the number would have been only 56 percent, which is not very high considering the fact that even the Democratic Whip, Jim Clyburn, voted 44 percent of the time with Boehner during that period. Basically, many votes in Congress are not as controversial as the public would think!*

For both claims above, we can verify their correctness using, for example, SQL queries over reliable, structured datasets available to the public. Database systems are good at verifying whether these claims are correct, but from the above discussion, it is obvious that assessing claim quality involves much more than testing correctness. Indeed, both claims above “check out” on the surface, but they present misleading views of the underlying data. The list of so-called “lies, d—ed lies, and statistics” goes on, in politics, sports, business, and even research—practically wherever numbers and data are involved.

While the lines of reasoning behind our assessment of the claims above are intuitive, deriving them requires considerable skill and effort. Not all users think as critically as we would hope. Not all users who are suspicious of a claim have the time or expertise to conduct further data analysis. How do we make this process of “fact-checking” easier and more effective?

This problem has many applications in domains where assessments and decisions are increasingly driven by data. *Computational journalism* [6, 7] is one domain with the most pressing need for better fact-checking techniques. With the movement towards accountability and transparency, the amount of data available to the public is ever increasing. Such data open up endless possibilities for empowering journalism’s watchdog function—to hold governments, corporations, and powerful individuals accountable to society. However, we are facing a widening divide between the growing amount of data on one hand, and the shrinking cadre of

[*]Y.W. and J.Y. are supported by NSF grants IIS-09-16027, IIS-13-20357. Y.W. and P.K.A are supported by NSF grant CCF-11-61359. P.K.A is additionally supported by NSF grants CCF-09-40671 and CCF-10-12254, by an ERDC contract W9132V-11-C-0003, by an ARO contract W911NF-13-P-0018, and by Grant 2012/229 from the U.S.-Israel Binational Science Foundation. C.L. is supported by NSF grants IIS-10-18865 and CCF-11-17369. J.Y. (2010) and C.L (2011, 2012) are also supported by HP Labs Innovation Research Awards. 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.

*Proceedings of the VLDB Endowment,* Vol. 7, No. 7
Copyright 2014 VLDB Endowment 2150-8097/14/03.

[1] This ad was in response to an earlier ad attacking Marshall for voting with Nancy Pelosi “almost 90 percent of the time,” which, not surprisingly, also tailored the claim in ways to further its own argument.

investigative journalists on the other. Computing is a key to bridge this divide. Automating fact-checking, as much as possible, will help data realize their potentials in serving the public.

**Challenges** To have any hope for automated fact-checking, can we formalize, mathematically, intuitions of seasoned fact-checkers when assessing claim quality (such as in Examples 1 and 2)? Do different claims require different intuitions and procedures to check? To what extent can fact-checking be approached in general ways?

Besides numerical measures of claim quality, *counterarguments* are critical in helping users, especially a non-expert public audience, understand why a claim has poor quality. As examples, we counter Giuliani's adoption claim with the argument that "*Comparing the adoption rates at the beginning and the end of this tenure... would have only yielded an increase of 17 percent*"; we counter the Marshall-Boehner vote correlation claim with the argument that "*... even the Democratic Whip, Jim Clyburn, voted 44 percent of the time with Boehner*" since 2007. Can we automatically generate such counterarguments?

In many situations, the original claims were stated in a vague way (often intentionally). Claims in both Examples 1 and 2 omit important details such as the exact time periods of comparison. Numbers are rounded (sometimes in ways that are "convenient"). Fact-checkers thus need to seek clarification from original claimants, but such prodding requires effort and credential, and often leads to delays. Can we automatically "reverse-engineer" vague claims to recover the omitted details?

**Our Contributions** To address the above challenges, we propose a fact-checking framework general enough to handle various types of claims and fact-checking tasks. The key insight is that a lot of fact-checking can be accomplished by "perturbing" the claims in interesting ways. Thus, we model a claim as a parameterized query over data, whose result would vary as we change its parameter setting, forming a (high-dimensional) surface which we call the ***q**uery **r**esponse **s**urface (QRS)*. In conjunction with QRS, we introduce the notions of relative *result strength*, which captures how a perturbation strengthens or weakens a claim, and of *parameter sensibility*, which captures how natural and relevant a perturbation is in the context of the claim being checked. To illustrate the modeling power of this framework:

- We show that many intuitive measures of claim qualities can be defined naturally. For example, a claim has low "robustness" if sensible perturbations of claim parameters lead to substantially weaker, or even opposite, conclusions.
- We show how to formulate fact-checking tasks—such as *finding counterarguments* and *reverse-engineering vague claims*, as discussed earlier—as optimization problems on the QRS.
- As concrete examples, we show how to use our framework to check *window aggregate comparison claims* (generalization of Giuliani's adoption claim in Example 1) and *time series similarity claims* (generalization of the Marshall-Boehner vote correlation claim in Example 2).

Besides the modeling challenges of fact-checking, we also address its computational challenges. Given a claim, it is costly (and sometimes infeasible) to compute the full QRS by evaluating a database query for every possible parameter setting. We introduce techniques at various levels for more efficient fact-checking:

- We propose three general "meta" algorithms that make different assumptions about claim properties and availability of low-level algorithmic building blocks. The baseline (Section 3.1) assumes no special building blocks but needs to examine all possible parameter settings. More efficient meta algorithms (Section 3.2) assume building blocks that enable enumeration of parameter settings in the order of sensibility. The most efficient meta algorithms (Section 3.3) use building blocks that enable a divide-and-conquer approach for searching through the parameter space. Together, these meta algorithms enable "pay-as-you-go" support for efficient fact-checking—new claim types can be supported with little effort upfront, but more efficient support can be enabled by plugging in instantiations of low-level building blocks without re-implementing the high-level algorithms.
- For window aggregate comparison claims and time series similarity claims, we develop efficient, specialized algorithmic building blocks that can be plugged into the meta algorithms above to enable fact-checking in real time for interactive users.

Finally, we experimentally validate the ability of our framework in modeling fact-checking tasks and producing meaningful results, as well as the efficiency of our computational techniques.

Limited by space, we focus on the computational challenges of finding counterarguments and reverse-engineering vague claims. Even with this focus, our rich model and diverse application domains give rise to numerous interesting problems with non-trivial solutions. It is only possible to sample some of them here, often at a high level; for comprehensive discussion and additional details, please refer to our technical report [28].

Before proceeding, we note that fact-checking in general requires a repertoire of techniques including but not limited to ours—such as how to find datasets relevant to given claims, how to translate claims to queries, how to check claims that cannot be readily derived from structured data, just to mention a few. These challenges are briefly discussed in [28], but are beyond the scope of this paper.

# 2 Modeling Framework for Fact-Checking

## 2.1 Components of the Modeling Framework

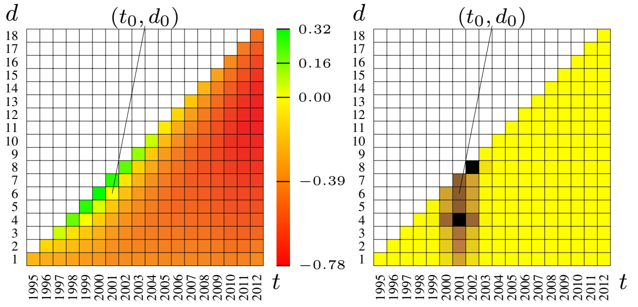
On a high level, we model the claim of interest as a parameterized query over a database, and consider the effect of perturbing its parameter settings on the query result. Besides the parameterized query, our model has two other components: 1) a measure of relative "strengths" of query results as we perturb query parameters, and 2) a measure of the "sensibility" of parameter settings, as not all perturbations make equal sense. In the following, we give additional intuition and formal definitions for our model.

**Parameterized Query Templates, QRS, and Claims** Let $q : \mathcal{P} \to \mathcal{R}$ denote a *parameterized query template*, where $\mathcal{P}$ is the *parameter space*, whose dimensionality is the number of parameters expected by $q$, and $\mathcal{R}$ is the *result space*, or the set of possible query results over the given database.[2] The ***q**uery **r**esponse **s**urface* of $q$, or *QRS* for short, is the "surface" $\{(p, q(p)) \mid p \in \mathcal{P}\}$ in $\mathcal{P} \times \mathcal{R}$.

A claim of type $q$ is specified by $\langle q, p, r \rangle$, where $p \in \mathcal{P}$ is the parameter setting used by the claim and $r \in \mathcal{R}$ is the result as stated by the claim. Obviously, if $r$ differs significantly from $q(p)$, the claim is *incorrect*. However, as motivated in Section 1, we are interested in the more challenging case where the claim is correct but nonetheless misleading. Doing so will involve exploring the QRS not only at the parameter setting $p$, but also around it.

For example, to check Giuliani's claim in Example 1, suppose we have a table `adopt(year, number)` of yearly adoption numbers.

[2] Here, we focus on perturbing query parameters, and assume the database $D$ to be given and fixed. In general, we can let $q$ additionally take $D$ as input, and consider the equally interesting question of perturbing data, or both data and query parameters; Section 7 briefly discusses this possibility.



(a) Relative strength of results (b) Sensibility of parameter settings

Figure 1: Perturbing $t$ (end of the second period) and $d$ (distance between periods) in Giuliani's claim while fixing $w = 6$ (length of periods). Note the constrain $t - d - w \geq 1988$; 1989 is when the data became available.

The parameterized query template here can be written in SQL, with parameters $w$ (length of the period being compared), $t$ (end of the second period), and $d$ (distance between the two periods):

```
SELECT after.total / before.total                    -- (Q1)
FROM (SELECT SUM(number) AS total FROM adopt
       WHERE year BETWEEN t-w-d+1 AND t-d) AS before,
      (SELECT SUM(number) AS total FROM adopt
       WHERE year BETWEEN AND t-w+1 AND t) AS after;
```

Giuliani's claim (after reverse-engineering) is specified by $\langle \texttt{Q1}, (w = 6, t = 2001, d = 6), 1.665\rangle$.

**Relative Strength of Results** To capture the effect of parameter perturbations on query results, we need a way to compare results. For example, if a perturbation in Giuliani's claim leads to a lower increase (or even decrease) in the total adoption number, this new result is "weaker" than the result of the claim. To this end, let $\mathsf{SR} : \mathcal{R} \times \mathcal{R} \to \mathbb{R}$ denote the *(relative) result strength function*: $\mathsf{SR}(r; r_0)$, where $r, r_0 \in \mathcal{R}$, returns the strength of $r$ relative to the *reference result* $r_0$. If $\mathsf{SR}(r; r_0)$ is positive (negative), $r$ is stronger (weaker, resp.) than $r_0$. We require that $\mathsf{SR}(r; r) = 0$. For example, we let $\mathsf{SR}(r; r_0) = r/r_0 - 1$ for Giuliani's claim.

Given a claim $\langle q, p_0, r_0\rangle$ to check, SR allows us to simplify the QRS of $q$ relative to $(p_0, r_0)$ into a surface $\{(p, \mathsf{SR}(q(p); r_0) \mid p \in \mathcal{P}\}$ in $\mathcal{R} \times \mathbb{R}$. We call this simplified surface the *relative result strength surface*. For example, Figure 1a illustrates this surface for Giuliani's adoption claim. Since the full three-dimensional parameter space is difficult to visualize, we fix $w$ to 6 and plot the surface over possible $t$ and $d$ values. Intuitively, we see that while some perturbations (near the diagonal, shown in greener colors) strengthen the original claim, the vast majority of the perturbations (shown in redder colors) weaken it. In particular, increasing $t$ and decreasing $d$ both lead to weaker claims. Thus, the surface leaves the overall impression that Giuliani's claim overstates the adoption rate increase. However, before we jump to conclusions, note that not all parameter settings are equally "sensible" perturbations; we discuss how to capture this notion next.

**Relative Sensibility of Parameter Settings** Some parameter perturbations are less "sensible" than others. For example, in Giuliani's claim, it makes little sense to compare periods with "unnatural" lengths (e.g., 13 years), or to compare periods "irrelevant" to Giuliani's term (e.g., periods in the 1970s). While "naturalness" of values is often an intrinsic property of the domain, "relevance" is often relative to the original claim (or its context). To capture overall sensibility, which is generally relative, we use either a *parameter sensibility function* or a *parameter sensibility relation*.

A *(relative) parameter sensibility function* $\mathsf{SP} : \mathcal{P} \times \mathcal{P} \to \mathbb{R}$ scores each parameter setting with respect to a *reference parameter setting*: $\mathsf{SP}(p; p_0)$ returns the *sensibility score* of $p \in \mathcal{P}$ with respect to $p_0 \in \mathcal{P}$. Higher scores imply more sensible settings. As an example, Figure 1b illustrates the relative sensibility of parameter settings for checking Giuliani's claim (again, we fix $w$ and vary only $t$ and $d$). Darker shades indicate higher sensibility. The interaction of naturalness and relevancy results in generally decaying sensibility scores around $(t_0, d_0) = (2001, 6)$ (because of relevancy), but with bumps when $d = 4$ and $d = 8$ (because of naturalness—the New York City mayor has 4-year terms). Intuitively, portions of the QRS over the high-sensibility regions of the parameter space are more "important" in checking the claim. See Section 4 for more details on SP for Giuliani's claim.

In some cases, there is no appropriate SP for ordering all parameter settings, but a weaker structure may exist on $\mathcal{P}$. A *(relative) parameter sensibility relation* $\preceq^{p_0}$, with respect to a *reference parameter setting* $p_0 \in \mathcal{P}$, is a partial order over $\mathcal{P}$: $p_1 \preceq^{p_0} p_2$ means $p_1$ is less sensible than or equally sensible as $p_2$ (relative to $p_0$). The sensibility relation $\preceq^{p_0}$ imposes less structure on $\mathcal{P}$ than the sensibility function SP—the latter actually implies a weak order (i.e., total order except ties) on $\mathcal{P}$. As an example, consider perturbing the Marshall-Boehner vote correlation claim by replacing Marshall with Clyburn. Intuitively, U.S. Representatives who are well recognizable to the public lead to more "natural" perturbations; on the other hand, "relevant" perturbations are Representatives who are even more liberal in ideology than Marshall (so as to counter the original claim's suggestion that Marshall is conservative). While it is difficult to totally order the discrete domain of Representatives, it makes sense to define a partial order based on their recognizability and ideology. See [28] for more details.

## 2.2 Formulating Fact-Checking Tasks

**Finding Counterarguments** Given original claim $\langle q, p_0, r_0\rangle$, a *counterargument* is a parameter setting $p$ such that $\mathsf{SR}(q(p); r_0) < 0$; i.e., it weakens the original claim. For example, Figure 1a shows counterarguments to Giuliani's claim in orange and red; they result in a lower percentage of increase (or even decrease) than what Giuliani claimed. Since there may be many counterarguments, we are most interested in those weakening the original claim significantly, and those obtained by highly sensible parameter perturbations. There is a trade-off between parameter sensibility and result strength: if we consider counterarguments with less sensible parameter perturbations, we might be able to find those that weaken the original claim more. Finding counterarguments thus involves bicriteria optimization. We define the following problems:

**(CA-$\tau_{\mathcal{R}}$)** Given original claim $\langle q, p_0, r_0\rangle$ and a *result strength threshold* $\tau_{\mathcal{R}} \leq 0$, find all $p \in \mathcal{P}$ with $\mathsf{SR}(q(p); r_0) < \tau_{\mathcal{R}}$ that are maximal with respect to $\preceq^{p_0}$; i.e., there exists no other $p' \in \mathcal{P}$ with $\mathsf{SR}(q(p'); r_0) < \tau_{\mathcal{R}}$ and $p' \succ^{p_0} p$.

**(CA-$\tau_{\mathcal{P}}$)** Beyond the partial order on $\mathcal{P}$, this problem requires the parameter sensibility function SP. The problem is to find, given original claim $\langle q, p_0, r_0\rangle$ and a *sensibility threshold* $\tau_{\mathcal{P}}$, all $p \in \mathcal{P}$ where $\mathsf{SP}(p; p_0) > \tau_{\mathcal{P}}$ and $\mathsf{SR}(q(p); r_0)$ is minimized.

For interactive exploration and situations when the choices of thresholds $\tau_{\mathcal{R}}$ and $\tau_{\mathcal{P}}$ are unclear, it is useful to enumerate Pareto-optimal counterarguments,[3] in descending order of parameter setting sensibility, until the desired counterargument is spotted. This problem is formulated below:

[3] More precisely, we say that a counterargument $p$ *dominates* a counterargument $p'$ if i) $\mathsf{SP}(p; p_0) \geq \mathsf{SP}(p'; p_0)$ (i.e., $p$ is more sensible than or equally sensible as $p'$); ii) $\mathsf{SR}(q(p); r_0) \leq \mathsf{SR}(q(p'); r_0)$ (i.e., $p$ weakens the original claim as much as or more than $p'$); and iii) inequality is strict for at least one of the above. A *Pareto-optimal* counterargument is one that is dominated by no counterarguments.

**(CA-po)** This problem requires the parameter sensibility function $\mathsf{SP}$. Given original claim $\langle q, p_0, r_0 \rangle$, find the $k$ Pareto-optimal counterarguments $p \in \mathcal{P}$ with the highest $\mathsf{SP}(p; p_0)$ values.

**Reverse-Engineering Vague Claims** As motivated in Section 1, many claims are not stated precisely or completely; e.g., Giuliani's adoption claim omits its parameter values and rounds its result value. We still represent a vague claim by $\langle q, p_0, r_0 \rangle$. However, $p_0$ and $r_0$ are interpreted differently from other problem settings. Here, $r_0$ may be an approximation of the actual result. For parameter values mentioned explicitly by the claim, $p_0$ sets them accordingly. On the other hand, for omitted parameters, $p_0$ sets them to "reasonable" values capturing the claim context. For example, Giuliani's adoption claim does not state the periods of comparison. We simply use 1993–1993 and 2001–2001 for $p_0$, i.e., $(w_0, t_0, d_0) = (1, 2001, 8)$, to capture the claim context that Giuliani was in office 1994–2001. Note that when picking $p_0$, we do not need to tweak it to make $q(p_0)$ match $r_0$; our problem formulation below will enable automatic reverse-engineering.

Any parameter setting is a candidate for a reverse-engineered claim. The reverse-engineering problem turns out to be very similar to the problem of finding counterarguments—we still seek a sensible parameter setting $p$ relative to $p_0$, but we want $p$ to lead to a result that is close to $r_0$ instead of weaker than it. The problem has three variants, RE-$\tau_{\mathcal{R}}$, RE-$\tau_{\mathcal{P}}$, and RE-po, analogous to their counterparts for CA. Because of limited space, we formally define only RE-$\tau_{\mathcal{R}}$ here.

**(RE-$\tau_{\mathcal{R}}$)** Given vague claim $\langle q, p_0, r_0 \rangle$ and a *result strength threshold* $\tau_{\mathcal{R}} > 0$, find all $p \in \mathcal{P}$ with $|\mathsf{SR}(q(p); r_0)| < \tau_{\mathcal{R}}$ that are maximal with respect to $\preceq^{p_0}$.

**Measuring Claim Quality** Quantifying claim quality requires a parameter sensibility function $\mathsf{SP} : \mathcal{P} \times \mathcal{P} \to \mathbb{R}$. Given the original parameter setting $p_0$, we further require $\mathsf{SP}(\cdot; p_0)$ to define a probability mass function (pmf),[4] or, if $\mathcal{P}$ is continuous, a probability density function (pdf). Consider a "random fact-checker," who randomly perturbs the parameter setting according to $\mathsf{SP}$; $\mathsf{SP}(p; p_0)$ represents the relative likelihood that the original parameter setting $p_0$ will be perturbed to $p$. This random fact-checker is more likely to pick settings that are "natural" and "relevant" (with respect to the original claim), as explained earlier.

Different quality measures make sense for claims of different types, or the same claim viewed from different perspectives. Because of limited space, we present only *robustness* here; see [28] for formal definitions of *fairness* and *uniqueness*. For simplicity of exposition, we assume that $\mathcal{P}$ is finite and discrete, and that $\mathsf{SP}$ is a pmf. Generalization to the continuous case is straightforward.

**(Robustness)** The *robustness* of a claim $\langle q, p_0, r_0 \rangle$ is

$$\exp\left(-\sum_{p \in \mathcal{P}} \mathsf{SP}(p; p_0) \cdot (\min(0, \mathsf{SR}(q(p); r_0)))^2\right).$$

Intuitively, robustness is computed from the mean squared deviation (from $r_0$) of perturbed claims generated by the random fact-checker. If a perturbed claim is stronger than the original, we consider the deviation to be 0. We use $\exp(-\cdot)$ to ensure that robustness falls in $(0, 1]$. Robustness of 1 means all perturbations result in stronger or equally strong claims; low robustness means the original claim can be easily weakened.

# 3 Algorithmic Framework for Fact-Checking

With the modeling framework in place, we now turn to our algorithmic framework for fact-checking. To make our techniques broadly applicable, we develop more generic algorithms than ones tailored to particular types of claims. However, more efficient algorithms are only possible by exploiting specific properties of the claims. To gain efficiency without sacrificing generality, we develop a series of "meta" algorithms:

- Our baseline algorithms (Section 3.1) assume only the availability of a generator function *GetP*, which returns, one at a time, all possible parameter perturbations of the claim to be checked.
- To avoid considering all possible parameter perturbations, our advanced algorithms assume more powerful building blocks: functions that generate parameter settings in decreasing sensibility order (Section 3.2), and those that support a divide-and-conquer approach for searching the parameter space (Section 3.3). Instantiations of such building blocks will be presented in Sections 4.2 and 5.2.
- Besides intelligent strategies for searching the parameter space, preprocessing of the input data can significantly reduce the cost of query evaluation for each parameter setting. Thus, we allow a customized data preprocessing function to be plugged in. Sections 4.2 and 5.2 presents examples of how plugged-in preprocessing helps with checking claims in Examples 1 and 2.

As motivated in Section 1, this approach enables an extensible system architecture with "pay-as-you-go" support for efficiency—new claim types can be supported with little configuration effort, but higher efficiency can be enabled by plugging in instantiations of low-level building blocks.

We tackle finding counterarguments (CA) and reverse-engineering (RE) here, and do not discuss how to compute various claim quality measures. Furthermore, we focus on CA below, because (unless otherwise noted) our meta algorithms for RE are straightforward adaptions of their counterparts for CA. Whenever their counterparts use $\mathsf{SR}(q(p); r_0)$, they simply use $|\mathsf{SR}(q(p); r_0)|$ instead.

## 3.1 Baseline Algorithms

The baseline algorithms assume nothing beyond the minimum required to define the problems. To find counterarguments, these algorithms simply call $\mathsf{GetP}$ to consider all possible parameter settings exhaustively. Because of limited space, we only describe the baseline algorithm for CA-po here; see [28] for other algorithms.

**(BaseCA-po)** Suppose the original claim is $\langle q, p_0, r_0 \rangle$. Given $k$, BaseCA-po solves CA-po by calling GetP to consider each possible parameter setting $p$. Finding the Pareto-optimal parameter settings amounts to the well-studied problem of computing maximal points in 2-d (sensibility and strength) [18, 5]. To avoid storing sensibility and strength for all possible parameter settings, BaseCA-po remembers only the Pareto-optimal subset $A$ of all parameter settings seen so far. We store $A$ in a search tree indexed by sensibility, which supports $O(\log t)$ update time, where $t$ denotes the maximum size reached by $A$ during execution. After considering all $p \in \mathcal{P}$, BaseCA-po returns the $k$ elements in $A$ with the highest sensibility.

The baseline algorithms work fine for small parameter spaces. However, their costs become prohibitive (especially for interactive use) when $|\mathcal{P}|$ is large, because they must exhaustively examine all possible parameter settings before returning any answer at all. More precisely, the baseline algorithms, for all three variants of CA, make $O(|\mathcal{P}|)$ calls to functions SR, SP, and GetP. Assuming each such call takes unit time, BaseCA-po takes $O(|\mathcal{P}| \log t)$ time and uses $O(t)$ space, where $t$ denotes the maximum size reached by $A$ (Pareto-optimal answer set found so far) during execution.

[4] When $\mathcal{P}$ is finite and discrete, given any definition for $\mathsf{SP}$, we can simply normalize it by $\sum_{p \in \mathcal{P}} \mathsf{SP}(p; p_0)$ to obtain a pmf.

## 3.2 Ordered Enumeration of Parameters

We now consider algorithms that take advantage of functions that enumerate, on demand, all parameter settings in non-increasing order of sensibility. When the parameter space is large, such functions enable us to focus first on exploring parts of it with the most sensible perturbations. There are three cases.

We start with the case (Section 3.2.1) when the parameter sensibility function $\mathsf{SP}(p; p_0)$ is defined, and an improved version of $\mathsf{GetP}$, which we denote by $\mathit{GetP}_{\downarrow}$, is available for generating parameter settings with high $\mathsf{SP}$ first.

The second case (Section 3.2.2) extends the first, and is common for multidimensional parameter spaces. Here, instead of requiring $\mathsf{GetP}_{\downarrow}$, we require, for each dimension $\imath$, a function $\mathit{GetP}_{\downarrow}^{d[\imath]}$ for enumerating the values in this dimension in order. We assume this order is consistent with the overall sensibility: i.e., given a parameter setting $p$, replacing its value for dimension $\imath$ with one that appears earlier in this order cannot decrease $\mathsf{SP}(p; p_0)$. We give a general procedure that uses the single-dimensional $\mathsf{GetP}_{\downarrow}^{\mathsf{d}[\imath]}$'s to implement a single multidimensional $\mathsf{GetP}_{\downarrow}$, so we can then apply the algorithms for the first case above. Giuliani's adoption claim can be handled with this approach, as we will see in Section 4—we need to specify how to enumerate values in each of the three dimensions $w$, $t$, and $d$ individually, but we need not define how to enumerate $(w, t, d)$ settings manually.

In the third case, again, no single $\mathsf{GetP}_{\downarrow}$ is given, but parameter settings can be compared according to multiple criteria. For each criterion $\jmath$, a function $\mathit{GetP}_{\downarrow}^{c[\jmath]}$ exists to enumerate parameter settings in order according to $\jmath$. The $\mathsf{GetP}_{\downarrow}^{\mathsf{c}[\jmath]}$'s together define a partial order $\preceq^{p_0}$ on $\mathcal{P}$. For example, the Marshall-Boehner vote correlation claim falls into this case when we permute the U.S. Representatives being compared—they can be ordered by either recognizability or ideology.

We now present our algorithms for the first two cases. Because of space limit, the details on the third case are presented in [28].

### 3.2.1 Algorithms based on $\mathsf{GetP}_{\downarrow}$

Suppose that given the parameter setting $p_0$ of the original claim, $\mathsf{GetP}_{\downarrow}(p_0)$ is available for generating parameter settings one at a time in non-increasing order of $\mathsf{SP}(p; p_0)$. The algorithms for finding counterarguments can be improved as follows.

**(EnumCA-po)** On a high level, EnumCA-po is similar to BaseCA-po. As EnumCA-po considers each parameter setting returned by $\mathsf{GetP}_{\downarrow}$, it also incrementally maintains the Pareto-optimal subset $A$ of parameter settings seen so far, as in BaseCA-po. Because $\mathsf{GetP}_{\downarrow}$ returns parameter settings in non-increasing order of sensibility, however, maintenance of $A$ becomes much easier. We can store $A$ as a list and update $A$ in $O(1)$ time, as in Kung et al.'s algorithm for 2-d skyline [18]. Furthermore, $|A|$ grows monotonically, so we can terminate once $|A|$ reaches $k$.

EnumCA-po runs in $O(s)$ steps, where $s \leq |\mathcal{P}|$ is the number of parameter settings it examines. It runs with $O(k)$ space, where $k$ is the desired number of results. Each step evaluates $q$ once and takes $O(1)$ time to update $A$. Thus, EnumCA-po outperforms BaseCA-po on all fronts: BaseCA-po runs in $O(|\mathcal{P}|)$ steps and with $O(t)$ space (recall that $t$ is the maximum size that $A$ can reach during BaseCA-po, which is not bounded by $k$ as in EnumCA-po), and each step takes $O(\log t)$ time to update $A$. The savings are significant in practice, because oftentimes $s \ll |\mathcal{P}|$ and $k \ll t$. Other algorithms can be similarly improved.

### 3.2.2 Enumerating Values in Each Dimension

Given the parameter setting $p_0$ of the original claim, suppose that for each dimension $\imath$ of the parameter space, a function $\mathsf{GetP}_{\downarrow}^{\mathsf{d}[\imath]}(p_0)$ is available for returning values in dimension $\imath$ in order, such that $\mathsf{SP}(p; p_0)$ is monotonically non-increasing with respect to the ordinal number of $p$'s value for dimension $\imath$ in the sequence returned by $\mathsf{GetP}_{\downarrow}^{\mathsf{d}[\imath]}(p_0)$. Since it is possible for some combinations of single-dimensional values to be an invalid parameter setting, we also assume a Boolean function $\mathsf{IsPValid}(p; p_0)$ that tests the validity of $p$. With these functions, we now show how to implement an overall $\mathsf{GetP}_{\downarrow}$ by combining these multiple $\mathsf{GetP}_{\downarrow}^{\mathsf{d}[\imath]}$'s.

We maintain the set of candidate parameter settings in a priority queue $Q$, whose priority is defined by sensibility. $Q$ is initialized with an entry whose components are obtained by calling each $\mathsf{GetP}_{\downarrow}^{\mathsf{d}[\imath]}(p_0)$ for the first time. Because of the monotonicity of $\mathsf{SP}$, this entry has the highest sensibility in $\mathcal{P}$. We always remove the highest-priority entry (say $p$) from $Q$ and, if it is valid, return it as the next parameter settings for $\mathsf{GetP}_{\downarrow}$. Suppose the entry removed is $p = (v_1, \ldots, v_{\dim(\mathcal{P})})$. Using it, we construct candidate parameter settings and insert them into $Q$. Each candidate parameter setting $p'$ is obtained by replacing one component $v_\imath$ of $p$ by the value that $\mathsf{GetP}_{\downarrow}^{\mathsf{d}[\imath]}(p_0)$ returns after $v_\imath$. We check that all predecessors $p''$ of $p'$ have been removed from $Q$, by comparing $\mathsf{SP}(p''; p_0)$ with $\mathsf{SP}(p; p_0)$. Hence, we avoid enumerating any $p$ multiple times.

Suppose the number of values per dimension is $O(\eta)$. The space complexity of this algorithm is $O(\eta^{\dim(\mathcal{P})-1})$, dominated by the priority queue; cached values from $\mathsf{GetP}_{\downarrow}^{\mathsf{d}[\cdot]}$ calls together take only $O(\dim(\mathcal{P})\eta)$. The time complexity is $O(\eta^{\dim(\mathcal{P})} \dim(\mathcal{P}) \log \eta)$, with $O(\log \eta^{\dim(\mathcal{P})-1})$ for each of the $O(\eta^{\dim(\mathcal{P})})$ parameter settings enumerated. If we exhaust the parameter space $\mathcal{P}$ using this algorithm, by enumerating all possible values along all dimensions, the time complexity can be higher than the baseline. However, this algorithm can stop execution early, enumerating only parameters in the "neighborhood" of $p_0$. Additional analysis in Section 4 will demonstrate this point analytically on a concrete problem instance.

## 3.3 Divide and Conquer on Parameter Space

For certain types of query templates, there exist more powerful algorithmic building blocks for efficiently finding parameter settings with the "best" result strengths within a region of the parameter space $\mathcal{P}$, without trying all parameter settings therein. For example, given a time series, with some preprocessing, it is possible to find the data point with the minimum value within a given time range; as we will show in Section 4.2, this building block allows us to find counterarguments for Giuliani's adoption claim quickly.

For these types of query templates, we develop algorithms that assume the availability of two functions: *DivP* is the *parameter space division function*, and *OptP* is the *parameter optimization function*. On a high level, these two functions together enable a divide-and-conquer approach: DivP divides $\mathcal{P}$ into "zones," and OptP returns the "best" parameter setting within each zone.

Formally, given a reference parameter setting $p_0$ and a parameter sensitivity threshold $\tau_{\mathcal{P}}$, $\mathsf{DivP}(p_0, \tau_{\mathcal{P}})$ returns a set of *zones*[5] in $\mathcal{P}$, whose union is exactly $\{p \in \mathcal{P} \mid \mathsf{SP}(p; p_0) > \tau_{\mathcal{P}}\}$, the subset of $\mathcal{P}$ with sensibility above $\tau_{\mathcal{P}}$. Given a zone $\psi$ and a reference result $r_0$, OptP has two variants: $\mathsf{OptP}_{-\infty}(r_0, \psi)$, for CA, returns $\arg\min_{p \in \psi} \mathsf{SR}(q(p); r_0)$, i.e., the parameter setting(s) in $\psi$ with

[5] We leave the definition for a "zone" of $\mathcal{P}$ up to DivP and OptP. The general guidelines for designing DivP and OptP are that each zone should contain a non-trivial number of parameter settings, and that OptP can work efficiently within each zone. The only requirement is that zones have compact descriptions, so they can be passed efficiently from DivP to OptP during processing. For example, a zone in $\mathbb{N}^3$ may be succinctly described as a convex region defined by a small number of inequalities. In contrast, an explicit list of member points would not be a good description.

minimum result strength relative to $r_0$; $\mathsf{OptP}_0(r_0, \psi)$, for RE, returns $\arg\min_{p \in \psi} |\mathsf{SR}(q(p); r_0)|$, i.e., the parameter setting(s) in $\psi$ whose result is closest to $r_0$. Using $\mathsf{DivP}$ and $\mathsf{OptP}_{-\infty}$, we now have the following algorithm for CA-$\tau_{\mathcal{P}}$.

**($\mathsf{DiCoCA}$-$\tau_{\mathcal{P}}$)** Given a parameter sensibility threshold $\tau_{\mathcal{P}}$, $\mathsf{DiCoCA}$-$\tau_{\mathcal{P}}$ simply calls $\mathsf{DivP}(p_0, \tau_{\mathcal{P}})$ to divide the set of parameter settings above the sensibility threshold (relative to $p_0$) into a set of zones. Then, for each zone $\psi$, $\mathsf{DiCoCA}$-$\tau_{\mathcal{P}}$ calls $\mathsf{OptP}(r_0, \psi)$ to find the best counterarguments. Finally, it returns the overall best counterarguments across all zones.

The main improvement of $\mathsf{DiCoCA}$-$\tau_{\mathcal{P}}$ over $\mathsf{EnumCA}$-$\tau_{\mathcal{P}}$ comes from the fact that the number of zones returned by $\mathsf{DivP}$ is often far less than the number of parameter settings with sensibility above $\tau_{\mathcal{P}}$. On the other hand, the cost of processing each zone is likely bigger than that of processing each parameter setting. Thus, the overall savings hinge on the efficiency of $\mathsf{OptP}$.

The divide phase avoids the $O(\dim(\mathcal{P}) \log \eta)$ factor in the time complexity of the earlier algorithm based on ordered enumeration. The conquer phase aims at finding the optimal parameter setting without calling $\mathsf{SR}$ for every possible parameter setting in the zone. The overall time complexity depends on the building blocks $\mathsf{DivP}$ and $\mathsf{OptP}$; the goal is to devise $\mathsf{DivP}$ and $\mathsf{OptP}$ to achieve $o(\eta^{\dim(\mathcal{P})})$ running time. We will see instantiations of $\mathsf{DivP}$ and $\mathsf{OptP}$ in Sections 4.2 and Section 5.2 that would enable this algorithm.

Algorithms for CA-$\tau_{\mathcal{R}}$ and CA-po build on top of $\mathsf{DiCoCA}$-$\tau_{\mathcal{P}}$ by invoking it multiple times with different "guesses" of values $\tau_{\mathcal{P}}$, while carefully avoiding repeated $\mathsf{DiCoCA}$-$\tau_{\mathcal{P}}$ calls with sensibility thresholds that are (effectively) the same. See [28] for details.

# 4 WAC: Window Aggregate Comparison Claims

With our modeling and algorithmic frameworks, we now show how to check a class of claims generalizing Giuliani's in Example 1.

## 4.1 Modeling WAC

**Parameterized Query Template** Here, the database is a sequence of positive numbers $x_1, x_2, \ldots, x_n$. A *window aggregate* with window length $w$ and endpoint $t$ computes $\sum_{i \in (t-w, t]} x_i$.[6] The *window aggregate comparison (WAC) query template* is the function

$$q(w, t, d) = \frac{\sum_{i \in (t-w, t]} x_i}{\sum_{i \in (t-d-w, t-d]} x_i}, \tag{1}$$

which compares two windows of the same length $w \in [1, n-1]$ ending at $t$ and $t-d$, respectively. We call $t \in [w+1, n]$ the *current time* and $d \in [1, t-w]$ the *lead*. Hence, the parameter space $\mathcal{P}$ is the set of points in $\mathbb{N}^3$ in a convex polytope, and the result space $\mathcal{R}$ is $\mathbb{R}^+$. The size of $\mathcal{P}$ for a data sequence of length $n$ is $O(n^3)$.

**Result Strength** Suppose the claim boasts an increase of aggregate value over time (which is the case for Giuliani's adoption claim). As mentioned in Section 2.2, we define the result strength function as $\mathsf{SR}(r; r_0) = r/r_0 - 1$. (On the other hand, if the claim boasts a decrease, e.g., "*crime rate is dropping*," we would replace $r/r_0$ with $r_0/r$ in $\mathsf{SR}(r; r_0)$.)

**Parameter Sensibility** We define parameter sensibility by dividing it into two components—"naturalness" $\mathsf{SP}^{\mathsf{nat}}(p)$ (independent of $p_0$) and "relevance" $\mathsf{SP}^{\mathsf{rel}}(p; p_0)$ (dependent on $p_0$):

$$\mathsf{SP}(p; p_0) \propto \mathsf{SP}^{\mathsf{nat}}(p) \cdot \mathsf{SP}^{\mathsf{rel}}(p; p_0). \tag{2}$$

We normalize $\mathsf{SP}(p; p_0)$ so that it is a pmf over $\mathcal{P}$ given $p_0$. Note that it also induces a weak order on $\mathcal{P}$.

First, consider naturalness. In general, for time series data, certain durations are more natural than others. For example, for monthly data, multiples of 12 (i.e., years) are more natural than multiples of 3 but not of 12 (i.e., quarters), who are in turn more natural than an integer not divisible by 3. For Giuliani's adoption claim over yearly adoption data, durations that are multiples of 4 are natural because the term of the New York City mayor is four years. Recognizing that a domain of time durations often has a periodic structure, we define naturalness for such a domain using a set of (usually a few, and often not disjoint) *levels* whose union is $\mathbb{N}$. Each level $\ell$ is specified by a pair $(\chi_\ell, \pi_\ell)$. Here, $\chi_\ell$ is the naturalness score associated with level $\ell$; $\pi_\ell \geq 1$ is an integral period that defines the domain values in level $\ell$ as $\mathbb{N}^{(\ell)} = \{v \in \mathbb{N} \mid \pi_\ell \text{ divides } v\}$. The naturalness score of a duration $v$ is given by $\max\{\chi_\ell \mid v \in \mathbb{N}^{(\ell)}\}$; i.e., the maximum score that $v$ is associated with.

For WAC, let $p = (w, t, d)$. Window length $w$ and lead $d$ are both durations, and contribute to the naturalness of $p$. We define

$$\mathsf{SP}^{\mathsf{nat}}(p) = \mathsf{SP}^{\mathsf{nat}}_w(w) \cdot \mathsf{SP}^{\mathsf{nat}}_d(d), \tag{3}$$

where $\mathsf{SP}^{\mathsf{nat}}_w$ and $\mathsf{SP}^{\mathsf{nat}}_d$ are naturalness scoring functions for the domains of $w$ and $d$ as discussed above. Specifically, for Giuliani's claim, we define naturalness for both domains using three levels $(1, 1)$, $(e, 4)$, $(e^2, 8)$. Here, periods 4 and 8 reflect the natural term lengths of New York City mayors; the choice of $e$ as bases is for convenience (when multiplied with a Gaussian relevance term). More sophisticated naturalness modeling is certainly possible, but we have found this definition to be adequate in our experiments.

Second, consider relevance. Generally speaking, the relevance of a parameter setting decreases with the magnitude of perturbation from the parameter setting of the original claim. For WAC, let $p_0 = (w_0, t_0, d_0)$ denote the original parameter setting. We define $\mathsf{SP}^{\mathsf{rel}}(p; p_0)$ as follows, which is essentially a Gaussian centered around $p_0$, with independent components and component-wise distances normalized by $\sigma_w$, $\sigma_t$, and $\sigma_d$:

$$\mathsf{SP}^{\mathsf{rel}}(p; p_0) = \mathsf{SP}^{\mathsf{rel}}_w(w; w_0) \cdot \mathsf{SP}^{\mathsf{rel}}_{ab}(t; t_0) \cdot \mathsf{SP}^{\mathsf{rel}}_d(d; d_0), \text{ where}$$
$$\mathsf{SP}^{\mathsf{rel}}_w(w; w_0) = e^{-\left(\frac{w-w_0}{\sigma_w}\right)^2}, \mathsf{SP}^{\mathsf{rel}}_{ab}(t; t_0) = e^{-\left(\frac{t-t_0}{\sigma_t}\right)^2}, \mathsf{SP}^{\mathsf{rel}}_d(d; d_0) = e^{-\left(\frac{d-d_0}{\sigma_d}\right)^2}. \tag{4}$$

Specifically, for Giuliani's claim, $(\sigma_w, \sigma_t, \sigma_d) = (5, 1, 10)$. Here, a small $\sigma_t$ penalizes perturbation in $t$, because its original setting reflects the end of Giuliani's term; a large $\sigma_d$ allows more flexibility in perturbing $d$ than $w$, as $w$ is more constrained by Giuliani's term.

Recall that Figure 1b illustrates the overall parameter sensibility function—the product of naturalness and relevance—for Giuliani's claim when fixing $w = 6$.

**Fact-Checking Tasks** The modeling above immediately enables the formulation of all problems in Section 2.2 related to finding counterarguments and reverse-engineering for WAC claims.

## 4.2 Algorithmic Building Blocks for WAC

### *4.2.1 Preprocessing to Speed up Queries*

Answering a WAC query given parameters $(w, t, d)$ normally takes $\Omega(w)$ time because it must examine all data in the windows being compared. By preprocessing the input sequence into a sequence of prefix sums, we can reduce the query time to $O(1)$. More specifically, given the input data $x_1, x_2, \ldots, x_n$, define $\bar{x}_i = \sum_{j=1}^{i} x_i$ for $i \in [1, n]$. With one pass over the input data, we compute and materialize the prefix sums using the recurrence $\bar{x}_i = \bar{x}_{i-1} + x_i$ starting from $\bar{x}_1 = x_1$. This preprocessing takes $O(n)$ time and results in an array of size $n$. Then, we can evaluate a WAC query with parameters $(w, t, d)$ as $\frac{\bar{x}_t - \bar{x}_{t-w}}{\bar{x}_{t-d} - \bar{x}_{t-d-w}}$ in $O(1)$ time.

[6] Here we assume sum; extensions to other common aggregation functions are straightforward.

### 4.2.2 Ordered Enumeration of Parameters

Enabling ordered enumeration of parameters for WAC is straightforward. As discussed in Section 3.2.2, for each of the three dimensions $w$, $t$, and $d$ of $\mathcal{P}$, we simply need to provide a function to enumerate its values in descending order of their contribution to SP; we also need to define the Boolean function IsPValid to test the validity of $(w, t, d)$ combinations. The algorithm described in Section 3.2.2 can then combine these functions automatically to provide ordered enumeration of full parameter settings.

We show how to implement $\mathsf{GetP}_{\downarrow}^{\mathsf{d}[w]}(w_0)$ for the dimension of window length $w$, where $w_0$ is from the original claim's parameter setting. Recall from Section 4.1 that $w$ contributes to both naturalness and relevance. For $w$ values within the same level of naturalness, enumerating them in the decreasing relevance order is straightforward, because they are found at increasing distance from $w_0$. To enumerate $w$ values in order of their overall contribution to SP, we perform enumeration across all levels of naturalness in parallel, using a priority queue to merge values from different levels into single stream.

The dimension of lead $d$ is analogous. The dimension of endpoint $t$ is simpler as it does not contribute to naturalness; we simply return $t$ values in increasing distance from $t_0$. Function IsPValid checks $t - d - w > 0$, to ensure that the earlier window falls completely within the input sequence.

To understand the complexity of ordered enumeration, we note that all parameter settings above a given sensibility fall within an ellipsoid centered at $p_0$; see Figure 2 for an illustration (a slice of the bounding ellipsoid is outlined in red; ignore the "zones" for now). Let $\tilde{r}$ denote the length of the longest semi-principal axis (measure by the number of possible values on it) of this ellipsoid; we call $\tilde{r}$ the *interesting solution radius*. For CA-$\tau_{\mathcal{P}}$, $\tilde{r}$ is determined by the given $\tau_{\mathcal{P}}$. For CA-$\tau_{\mathcal{R}}$ (or CA-po), $\tilde{r}$ is determined by the "effective" $\tau_{\mathcal{P}}$, i.e., the sensibility of the answer (or the lowest sensibility in the answer set, respectively). The same analysis applies to the variants of RE. Using the results in Section 3.2.2, the time and space complexities of ordered enumeration for WAC are $O(\tilde{r}^3 \log \tilde{r})$ and $O(\tilde{r}^2)$, respectively. In the worst case, $\tilde{r} = \Theta(n)$. However, in practice, a counterargument or reverse-engineered claim is often close to $p_0$, so $\tilde{r} \ll n$, and ordered enumeration will run much faster than the $O(n^3)$ baseline.

### 4.2.3 Divide and Conquer on Parameter Space

We now show how to enable an efficient divide-and-conquer approach to checking WAC claims, by defining functions DivP and OptP (Section 3.3). The subset of $\mathcal{P}$ above sensibility threshold $\tau_{\mathcal{P}}$ is a set of 3-d grid points. Roughly speaking, our approach "slices" this subset using planes perpendicular to the $w$ axis, and computes the best answer within each slice.

**Divide** In more detail, the parameter space division function DivP works as follows. For each possible window length $w$, for each naturalness level $\ell$ in the domain of $d$ specified by $(\chi_\ell^d, \pi_\ell^d)$, and then for each $h \in [0, \pi_\ell^d)$, we define a zone $\psi_{w,\ell,h}$ using the constraints below (note that $w$ is fixed):

$$t \le n \ \wedge \ t - d \ge w; \tag{5}$$

$$d \bmod \pi_\ell^d = 0; \tag{6}$$

$$\begin{aligned} &\left(\tfrac{t-t_0}{\sigma_t}\right)^2 + \left(\tfrac{d-d_0}{\sigma_d}\right)^2 < -\ln \tau, \text{ where} \\ &\tau = \tau_{\mathcal{P}} / \left(\mathsf{SP}_w^{\mathsf{nat}}(w) \cdot \mathsf{SP}_w^{\mathsf{rel}}(w; w_0) \cdot \chi_\ell^d\right); \end{aligned} \tag{7}$$

$$t \bmod \pi_\ell^d = h. \tag{8}$$

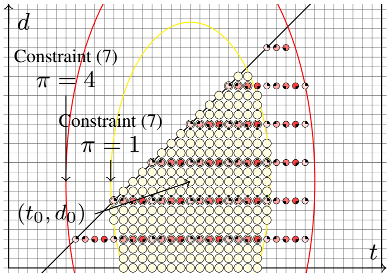


Figure 2: Illustration of zones $\psi_{w,\ell,h}$ (for a fixed $w$) dividing the parameter space of WAC claims. The yellow dots (○) belong to the zone for the level with period 1 and the lowest naturalness; the four types of red dots (◔◕●●), distinguished by their color saturation and orientation, belong to the four zones (with $h = 0, 1, 2, 3$) that make up the level with period 4 and higher naturalness. We shown only two levels here for simplicity.

Constraint (5) is implied by the problem definition. Constraint (6) says that $d$ belongs to the current level $\ell$ being considered. Constraint (7) enforces that the overall sensibility is above $\tau_{\mathcal{P}}$. As Figure 2 illustrates, the union of all zones $\psi_{w,\ell,h}$ with the same $w$ and $\ell$ is a uniform $(t, d)$ grid in the plane (with fixed $w$) bounded by a convex region; the spacing of this grid is 1 along the $t$-dimension and $\pi_\ell^d$ along the $d$-dimension. Constraint (8) further divides this grid into $\pi_\ell^d$ zones (by $h$), where each zone is a subgrid with spacing of $\pi_\ell^d$ along the $t$-dimension.

Here is some intuition of why a zone is defined as above. Consider a zone $\psi_{w,\ell,h}$. First, with $w$ fixed, $q(w, t, d)$ can be written as $y_t / y_{t-d}$ ($y$ will be formally defined below). Given this decomposition of $q$, in order to maximize/minimize $q(w, t, d)$, we only need to maximize/minimize $y_t$ and minimize/maximize $y_{t-d}$, which is a 1-d problem. However, not all combinations of $(t, d)$ are valid according to the semantics of WAC claim. Constraint (8) ensures that for any $t$, values of $d$ for valid $(t, d)$ combinations are contiguous in $\psi_{w,\ell,h}$, which is convenient for optimization as shown below.

**Conquer: CA** Next, we show how to conquer each zone returned by DivP. For the problem of finding counterarguments, we define $\mathsf{OptP}_{-\infty}$ for WAC (see [28] for pseudocode) as follows. Note that each zone $\psi_{w,\ell,h}$ is associated with an equally-spaced subsequence of time points $I = \{i \mid i \bmod \pi_\ell^d = h \wedge i^{\min} \le i \le i^{\max}\}$, where $i^{\min}$ and $i^{\max}$ are derived from the constraints on $t$ and $d$ imposed by the zone. We compute the window aggregate result (with window length $w$) for each of these time points, obtaining a sequence $Y = \{y_i \mid i \in I\}$, where $y_i = \sum_{j \in (i-w, i]} x_j = \bar{x}_i - \bar{x}_{i-w}$. Recall that the $\bar{x}_i$'s are precomputed prefix sums, so computing $Y$ takes $O(|Y|)$ time. Every $(t, d)$ setting in zone $\psi_{w,\ell,h}$ corresponds to a pair of time points in $I$, namely $(i_1, i_2) = (t - d, t)$; we call such pairs *valid*. Note that $\mathsf{SR}((w, t, d); r_0) = \frac{y_{i_2}}{y_{i_1}} \cdot \frac{1}{r_0} - 1$.

Thus, to minimize $\mathsf{SR}((w, t, d); r_0)$ within the zone, we look for a valid $(i_1, i_2)$ pair that minimizes $\frac{y_{i_2}}{y_{i_1}}$. To this end, for each valid $i_2$ value, we determine the range of valid $i_1$ values within which to maximize $y_{i_1}$. Because of the convexity of the zone, this range covers a contiguous subsequence of $\{y_i\}$. Hence, given $i_2$, the maximization problem reduces to a range-maximum query over a (static) sequence, a well-studied problem. The sequence $Y$ can be preprocessed in $O(|Y|)$ time into a linear-size data structure, so that any range-maximum query can be answered in $O(1)$ time [13, 4, 10].[7]

[7] We actually use a simple implementation based on Tarjan's offline LCA algorithm [24]. It has linear space and preprocessing time, but $O(\alpha(|Y|))$ time per range-maximum query (where $\alpha(\cdot)$ is the inverse Ackermann function), which already provides adequate performance.

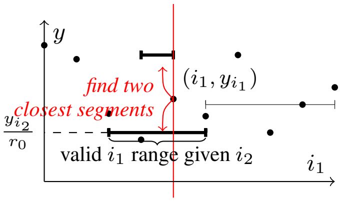


Figure 3: Illustration of the sweep line procedure used by OptP$_0$ for reverse-engineering WAC claims. Points corresponding to $Y$ are drawn as black dots. Segments currently intersected by the (red) sweepline are drawn using thick lines.

We use the same notion of "interesting solution radius" $\tilde{r}$ introduced in Section 4.2.2 to analyze the complexity of divide-and-conquer for WAC with the DivP and OptP$_{-\infty}$ described here. The time complexity of finding counterarguments for WAC is improved to $O(\tilde{r}^2)$—with the space divided into $O(\tilde{r})$ slices, each taking $O(\tilde{r})$ time to solve—compared with $O(\tilde{r}^3 \log \tilde{r})$ for ordered enumeration. The space complexity is improved to $O(\tilde{r})$, compared with $O(\tilde{r}^2)$ for ordered enumeration.

**Conquer: RE** For reverse-engineering WAC claims, we define OptP$_0$ as follows. Given a zone $\psi_{w,\ell,h}$, we derive $I$ and precompute the sequence $Y = \{y_i \mid i \in I\}$ as in OptP$_{-\infty}$, such that each $(t, d)$ setting in the zone corresponds to a pair of time points in $I$, namely $(i_1, i_2) = (t - d, t)$. However, instead of minimizing $\frac{y_{i_2}}{y_{i_1}} \cdot \frac{1}{r_0} - 1$[8] over all valid $(i_1, i_2)$ pairs as in OptP$_{-\infty}$, we want to minimize the absolute result strength difference $|\frac{y_{i_2}}{y_{i_1}} \cdot \frac{1}{r_0} - 1|$, or, equivalently, $|y_{i_1} - y_{i_2} \cdot \frac{1}{r_0}|$.

Given a valid $i_2$ value, we can determine the range of valid $i_1$ values associated with $i_2$, as in OptP$_{-\infty}$. Recall that OptP$_{-\infty}$ preprocesses $Y$, issues a query for each $i_2$ to find the best $i_1$ in the associated range, and then picks the overall best $(i_1, i_2)$ pair. Here, however, we find the overall best $(i_1, i_2)$ using a sweep line procedure, which essentially considers the $i_1$ ranges associated with all valid $i_2$'s in batch. To illustrate, let us map the sequence $Y = \{y_i \mid i \in I\}$ to a set of points $\{(i, y_i) \mid i \in I\}$ in 2-d as shown in Figure 3. For each valid $i_2$ value and its associated $i_1$ range, we draw a horizontal line segment spanning the $i_1$ range, at height $y_{i_2}/r_0$. It is easy to see that $i_1$ value that minimizes $|\frac{y_{i_2}}{y_{i_1}}/r_0 - 1|$ within this range corresponds to either the closest point above the segment or the closest point below the segment (with the constraint that the segment contains the projections of these points). Hence, the problem reduces to that of finding such closest point-segment pairs. To solve this problem, we sort the segments by the horizontal coordinates of their endpoints. Using a vertical sweep line, we consider the segments and points together in order. During the sweep, we incrementally maintain the set of segments intersected by the sweep line in a 1-d search tree (e.g., B-tree) keyed on their heights. For each incoming point $(i, y_i)$, we probe the search tree with $y_i$ for the active segments with heights closest to $y_i$ (above and below). We keep track of the best point-segment pair (i.e., one with the smallest $|\frac{y_{i_2}}{y_{i_1}}/r_0 - 1|$) seen so far during the sweep, and return it at the end of the sweep.

Preparing the segments for the sweep takes $O(|Y| \log |Y|)$ time. The sweep takes $O(|Y|)$ steps, each taking $O(\log |Y|)$ time. Therefore, OptP$_0$ takes $O(|Y| \log |Y|)$ time and $O(|Y|)$ space.

Similar to finding counterarguments earlier, the parameter space is divided into $O(\tilde{r})$ slices by DivP, each taking $O(\tilde{r} \log \tilde{r})$ time to solve by OptP$_0$. Thus, the overall time complexity is $O(\tilde{r}^2 \log \tilde{r})$. The space requirement is again linear.

## 5 TSS: Time Series Similarity Claims

We now turn to a class of claims generalizing Example 2.

### 5.1 Modeling TSS

**Parameterized Query Template** Here the database contains information about $m$ entities identified as $1, 2, \ldots, m$. Each entity $i$ is associated with a time series $X_i = \{x_{i,1}, x_{i,2}, \ldots, x_{i,n}\}$. A function $\text{sim}_T(X_i, X_j)$, where $T \subseteq [1, n]$ and $i, j \in [1, m]$, computes the similarity between two time series $\{x_{i,t} \in X_i \mid t \in T\}$ and $\{x_{j,t} \in X_j \mid t \in T\}$, i.e., $X_i$ and $X_j$ restricted to the subset of timesteps in $T$. The *time series similarity (TSS) query template* is the function $q(u, v, a, b) = \text{sim}_{[a,b]}(X_u, X_v)$, which compares *source entity* $u$ against *target entity* $v$ ($u \neq v$) over the time period $[a, b] \subseteq [1, n]$. Entity $v$ is typically well recognizable to the audience, and serves as the target of comparison in order to claim something desirable about $u$.

For example, in the Marshall-Boehner claim (reverse-engineered) of Example 2, $v$ is Boehner and $u$ is Marshall; $a$ corresponds to January 2007, and $b$ corresponds to October 14, 2010, the time when the claim was made. Each vote is one of *yea*, *nay*, *present but not voting*, and *absent*. The similarity between two Representatives over a period is computed as the number of times they both voted *yea* or *nay*, divided by the number of times neither is *absent*.

**Result Strength and Parameter Sensibility** We can investigate a TSS claim in multiple ways by parameter perturbation—changing the period of comparison, replacing the entities being compared, or both—which lead to multiple useful problem formulations. In many cases, it makes sense to perturb some instead of all parameters; doing so gives cleaner problem formulations and solutions that are easier to interpret. In general, different problem formulations call for different setups for parameter sensibility and result strength. Here, we focus on the problem of finding counterargument by perturbing the comparison period. Two other problems—finding counterarguments by perturbing entities, and reverse-engineering vague TSS claims—are discussed in [28].

Fixing $u$ and $v$ to those in the original claim, we can consider counterarguments obtained by perturbing $a$ and $b$. We call this problem *TSS-CA*$_{ab}$. Suppose higher similarity strengths the claim (which is the case for the Marshall-Boehner claim). We define the result strength function as $\mathsf{SR}(r; r_0) = r - r_0$. For parameter sensibility, we define the sensibility of $(a, b)$ relative to $(a_0, b_0)$ as the product of naturalness and relevance, as in Eq. (2).

In more detail, naturalness stems from $a$. In the vote correlation example, values of $a$ that correspond to the beginning of some session of Congress are the most natural. As with the naturalness of durations discussed in Section 4.1, we define naturalness of time points using a set of (not necessarily disjoint) levels whose union is $\mathbb{N}$. The relevance of $(a, b)$ relative to $(a_0, b_0)$ decreases with the distance between them, and is defined analogously to WAC claims in Eq. (4), Section 4.1. Specifically, for the Marshall-Boehner claim, $(\sigma_a, \sigma_b) = (1000, 1)$. We use a small $\sigma_b$ to penalize perturbation in $b$, because its original setting reflects the time of the claim. With the definitions above, we obtain the three variants of the problem of finding counterarguments in Section 2.2, for perturbations of the comparison time period.

### 5.2 Algorithmic Building Blocks for TSS

We now discuss how to instantiate the meta algorithms in Section 3 for TSS. We focus on TSS-CA$_{ab}$; see [28] for approaches to other problems for TSS. Since ordered enumeration for TSS-CA$_{ab}$ is a 2-d analogy to that of WAC-CA, we omit its discussion and focus on precomputation and divide-and-conquer below.

[8] Recall that if the original claim boasts a decrease, we would define SR as $r_0 \cdot \frac{y_{i_1}}{y_{i_2}} - 1$, but the same algorithm applies.

### 5.2.1 Preprocessing to Speed up Queries

For TSS-CA$_{ab}$, we are given entities $u$ and $v$ but need to permute the time period $[a, b]$. We preprocess the two time series $X_u = \{x_{u,t}\}$ and $X_v = \{x_{v,t}\}$ so that queries with any $(a, b)$ setting can be quickly answered. At the very least, the two time series can be materialized separately from the input data and further indexed by time. If the similarity function $\mathrm{sim}_{[a,b]}(X_u, X_v)$ is a *distributive or algebraic aggregate* [12] over the set $\{\mathrm{sim}_{[t,t]}(X_u, X_v) \mid t \in [a, b]\}$ of point-wise similarities—which is the case for vote correlation claims—we can do better. In this case, using an idea similar to that of prefix sums in Section 4.2.1, we define $\bar{s}_i$ as the number of times during $[1, i]$ that $u$ and $v$ agreed (i.e., they both voted *yea* or *nay*), and $\bar{c}_i$ as the number of times $u$ and $v$ both voted (i.e., neither was *absent*), where $i \in [0, n]$. We can incrementally compute the $\bar{s}_i$'s and $\bar{c}_i$'s with one pass over $X_u$ and $X_v$, starting with $\bar{s}_0 = \bar{c}_0 = 0$. Then, with materialized $\bar{s}_i$'s and $\bar{c}_i$'s, we can compute each query $\mathrm{sim}_{[a,b]}(X_u, X_v) = \frac{\bar{s}_b - \bar{s}_{a-1}}{\bar{c}_b - \bar{c}_{a-1}}$ in $O(1)$ time.

### 5.2.2 Divide and Conquer on Parameter Space

We now describe how to enable the divide-and-conquer approach for TSS-CA$_{ab}$. On a high level, the approaches are similar to those for WAC claims described in Section 4.2.3, but the details and underlying algorithmic challenges differ.

The subset of $(a, b)$ parameter settings above sensibility threshold $\tau_{\mathcal{P}}$ is a set of 2-d grid points. This subset is analogous to a slice of the 3-d grid points (with $w$ fixed) in Section 4.2.3, but further division into zones is simpler in this case. For each naturalness level $\ell$ in the domain of $a$ specified by $(\chi_\ell, \lambda_\ell)$, we let $\mathsf{DivP}$ return zone $\psi_\ell$ defined by the constraints below:

$$1 \le a \le b \le n; \tag{9}$$

$$\lambda_\ell(a) = \mathtt{true}; \tag{10}$$

$$\left(\frac{a-a_0}{\sigma_a}\right)^2 + \left(\frac{b-b_0}{\sigma_b}\right)^2 < -\ln(\tau_{\mathcal{P}}/\chi_\ell). \tag{11}$$

In the case of vote correlation claims, there are simply two zones: the low-naturalness zone is the set of grid points within a clipped ellipse defined by Constraints (9) and (11); the high-naturalness zone is a subset of the gird points with $a \in \mathbb{N}^{\mathsf{begin}}$.

We define $\mathsf{OptP}_{\infty}$ for TSS-CA$_{ab}$ as follows. Given zone $\psi_\ell$, we consider each valid $a$ value in turn. For each $a$, we determine the range of valid $b$ values (which is contiguous) using Constraints (9) and (11). Finding the $b$ value within this range that minimizes $\mathsf{SR}((a, b); (a_0, b_0))$ amounts to minimizing $\frac{\bar{s}_b - \bar{s}_{a-1}}{\bar{c}_b - \bar{c}_{a-1}}$.

We solve this minimization problem by a binary search on the lower boundary of the convex hull (*lower hull* for short) of the set of 2-d points $\{(\bar{c}_i, \bar{s}_i)\}$, where $i$ falls within the given range. Figure 4 illustrates the intuition behind this procedure. The set of points $O = \{o_i = (\bar{c}_i, \bar{s}_i) \mid i \in [1, n]\}$ form a non-decreasing staircase in 2-d. Given $a$ and the subset $O_a$ of the points in the corresponding $b$ range (which lies somewhere to the right of $o_{a-1}$), the point $o_{b^\star} \in O_a$ we are seeking minimizes the slope of the line $L_{a-1,b^\star}$ (i.e., connecting points $o_{a-1}$ and $o_{b^\star}$). Clearly, $o_{b^\star}$ must be on the lower hull of $O_a$, denoted $\mathsf{LH}(O_a)$. Furthermore, for each point $o_b \in \mathsf{LH}(O_a)$, consider the points $o_{b^-}$ and $o_{b^+}$ immediately to its left and right (respectively) in $\mathsf{LH}(O_a)$. The point we are seeking forms a *tangent* to the hull from point $o_{a-1}$; i.e., it is the only point for which both $o_{b^-}$ and $o_{b^+}$ lie above line $L_{a-1,b}$. If $o_{b^-}$ lies above $L_{a-1,b}$ and $o_{b^+}$ lies below $L_{a-1,b}$, then $o_{b^\star}$ is to the right of $o_b$; if $o_{b^-}$ lies below $L_{a-1,b}$ and $o_{b^+}$ lies above $L_{a-1,b}$, then $o_{b^\star}$ is to the left of $o_b$. This observation allows us to find $o_{b^\star}$ with a binary search in $\mathsf{LH}(O_a)$—given a guess $o_b$, the positions of $o_{b^-}$ and $o_{b^+}$ relative to $L_{a-1,b}$ tell us whether to search further to the left or right of $o_b$.

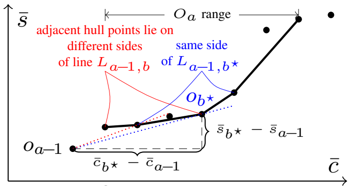

$\bar{s}$
$O_a$ range
adjacent hull points lie on different sides of line $L_{a-1,b}$
same side of $L_{a-1,b^\star}$
$o_{b^\star}$
$\bar{s}_{b^\star} - \bar{s}_{a-1}$
$o_{a-1}$
$\bar{c}_{b^\star} - \bar{c}_{a-1}$
$\bar{c}$

Figure 4: Illustration of $\mathsf{OptP}_{\infty}$ for finding counterarguments to TSS claims. Points $\{(\bar{c}_i, \bar{s}_i) \mid i \in [1, n]\}$ are drawn as black dots. The lower hull for the set $O_a$ of points is drawn as thick lines.

Let $G$ denote the set of valid $b$ ranges, each associated with a possible $a$ value. What remains to be shown is how to compute the lower hulls for all ranges in $G$ efficiently. Instead of computing them independently, which would take a total of $O(n|G|)$ time, we batch-compute them in $O(n)$ time as follows.

We divide $G$ into at most three batches, such that the ranges in each batch, when ordered non-decreasingly by their right endpoints, are either ordered non-increasingly or non-decreasingly by their left endpoints. More precisely, each batch of $m$ possible $a$ values and associated ranges can be represented as a list $\{(a_j, [l_j, r_j]) \mid j \in [1, m]\}$, where $r_1 \le r_2 \le \cdots \le r_m$, and either $l_1 \ge l_2 \ge \cdots \ge l_m$ or $l_1 \le l_2 \le \cdots \le l_m$. These lists can be generated simply by considering all possible $a$ values in order; no further sorting is needed. We process the ranges in each list (batch) in order, incrementally maintaining the lower hull when moving from one range to the next. There are two cases here.

*Case I.* The list satisfies $l_1 \ge l_2 \ge \cdots \ge l_m$; i.e., $O_{a_j}$'s range expands in both directions as $j$ increases. In the $j$-th iteration, we update the hull from $\mathsf{LH}(O_{a_{j-1}})$ to $\mathsf{LH}(O_{a_j})$, by incrementally adding points to the immediate left of $O_{a_{j-1}}$ from right to left (i.e., $o_{l_{j-1}-1}, o_{l_{j-1}-2}, \ldots, o_{l_j}$), as well as points to the immediate right of $O_{a_{j-1}}$ from left to right (i.e., $o_{r_{j-1}+1}, o_{r_{j-1}+2}, \ldots, o_{r_j}$), applying the sweep line method for convex hull computation [3] in both directions. Although adding each point may require deleting multiple points closest to it in the hull, the total time for the entire batch is only $O(n)$ because each point can be deleted from the evolving hull at most once.

*Case II.* The list satisfies $l_1 \le l_2 \le \cdots \le l_m$; i.e., both endpoints of $O_{a_j}$'s range move right as $j$ increases. To update the hull from $\mathsf{LH}(O_{a_{j-1}})$ to $\mathsf{LH}(O_{a_j})$, we first incrementally add points to the immediate right of $O_{a_{j-1}}$, and then delete points to the immediate left of $O_{a_j}$. Addition of points is done in the same way as in Case I. Deletion of points requires more care to avoid excessive recomputation of the parts of the lower hull. To aid deletion processing, we maintain an "anchor point" $o_h$, with the invariant that $o_h \in \mathsf{LH}(O_{a_j})$ at the end of the $j$-th iteration; furthermore, for every point $o_i$ where $i \in [l_j, h)$, we remember $\mathsf{next}(o_i)$ as the point next to $o_i$ on $\mathsf{LH}(\{o_i, o_{i+1}, \ldots, o_h\})$. Before processing the first range in the batch, we initialize $o_h$ to be $o_{l_1}$ (i.e., for a dummy 0-th range $[l_0, r_0] = [l_1, l_1]$). During the processing of the $j$-th range $[l_j, r_j]$ in the batch, after processing additions to obtain $H = \mathsf{LH}(\{o_{l_{j-1}}, o_{l_{j-1}+1}, \ldots, o_{r_j}\})$, we update the anchor point and the $\mathsf{next}(\cdot)$ values, and compute $\mathsf{LH}(O_{a_j})$ as follows (also shown in Figure 5):

(i) First, suppose $l_j \le h$, i.e., the anchor point remains in range. In this case, we discard the points in $H$ to the left of $o_{l_j}$, as well as any $\mathsf{next}(\cdot)$ values for points in $O$ to the left of $o_{l_j}$. Then, we add $o_{l_j}$, $\mathsf{next}(o_{l_j})$, $\mathsf{next}(\mathsf{next}(o_{l_j}))$, etc. to $H$ until we encounter a point in this series that is already in $H$. We have $\mathsf{LH}(O_{a_j}) = H$.

(ii) Suppose $l_j > h$, i.e., the range is moving past the anchor point. In this case, we discard the points in $H$ to the left of $o_{l_j}$, as well as all current $\mathsf{next}(\cdot)$ values. We let the new anchor point $o_h$ be

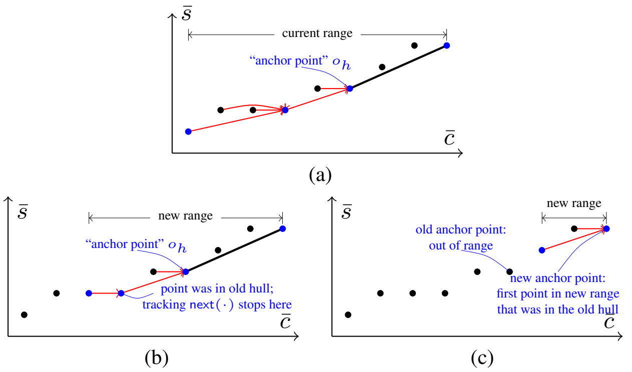


Figure 5: Update the lower hull w.r.t. deletion of leftmost points in the current range. Lower hull points are in blue. (a) before deletion; (b) case (i) anchor point remains in rage after deletion; (c) case (ii) anchor point goes out of range after deletion.

the first point in $H$ at or to the right of $l_j$. Then, to compute $\mathsf{LH}(O_{a_j})$, we start with $H$ and incrementally consider adding to its left points $o_{h-1}, o_{h-2}, \ldots, o_{l_j}$, in the same way as in Case I. This process also computes $\mathsf{LH}(\{o_i, o_{i+1}, \ldots, o_h\})$ for each $i \in [l_j, h)$ (in reverse order), allowing us to remember each $\mathsf{next}(o_i)$.

Note that each point $o_i \in O$ can be associated with at most one anchor point, so $\mathsf{next}(o_i)$ is only computed once (and in $O(1)$ time amortized) in (ii) above. Moreover, $\mathsf{next}(o_i)$ can be examined at most once in (i) above, because the series of additions to $H$ stops as soon as a point is already in $H$. Therefore, the total time for processing the entire batch is still $O(n)$.

Overall, it takes $O(n)$ time to incrementally construct the lower hull for each range of $G$, with ordered endpoints of the ranges. For each range $[l_j, r_j]$ of $G$, a binary search on $\mathsf{LH}(P_{a_j})$ is performed to find the optimal point in $P_{a_j}$ with respect to $p_{a_j}$, using $O(\log n)$ time. The total time and space complexities are $O(n \log n)$ and $O(n)$, respectively. More precisely, if we bound the size of $G$ using $O(\tilde{r})$ instead of $O(n)$, the time and space complexities become $O(\tilde{r} \log \tilde{r})$ and $O(\tilde{r})$.

# 6 Experiments

We begin with proof-of-concept experiments that apply our QRS framework to Examples 1 and 2, and that illustrate the usefulness of our results. Next, we demonstrate the efficiency and scalability of our algorithms, showing how enumeration and divide-and-conquer approaches lead to faster running times for interactive fact-checking.

All algorithms are implemented in C++. All experiments ran on a machine with the Intel Core i7-2600 3.4GHz processor and 7.8GB of memory. Besides the small adoption dataset for the New York City, we use the following datasets: ***UNEMP***[9] records the US monthly unemployment rate for 782 months from January 1948 to February 2013; ***AUTO***[10] contains daily auto accident statistics in Texas from 2003 to 2011 (with 3287 data points); ***VOTE***[11] contains 22.32 million votes cast by 12,572 members of the US Congress from May 1789 to April 2013.

## 6.1 Proof of Concept

**Giuliani's Adoption Claim** We first use our technique to reverse-engineer Giuliani's vague adoption claim in Example 1. Recall the model in Section 4.1. As discussed in Section 2.2, we set $p_0 = (1, 2001, 8)$, representing the two one-year windows 1993–1993 and 2001–2001 that are eight years apart. This captures the claim context that Giuliani served the 8-year term of Mayor of NYC during 1994-2001. Since the claim stated a "65 to 70 percent" increase, we set $r_0$ to be the geometric mean of 1.65 and 1.70. We ran our algorithm for RE-po; the top two answers (ordered by sensibility) were $(4, 2001, 7)$ and $(6, 2001, 6)$. The second (comparing 1990–1995 and 1996–2001) is exactly what Giuliani's claim used. The first one (comparing 1991–1994 and 1998–2001) also gives "65 to 70 percent" increase, and is arguably more sensible because it compares 4-year periods (term for mayor).

Next, given Giuliani's claim, reverse-engineered as $(6, 2001, 6)$, we ran our algorithm for CA-po to find counterarguments. The top answer was $(4, 2001, 4)$, which compares 1994–1997 and 1998–2001, i.e., Giuliani's first and second 4-year terms. This counterargument leads to $1\%$ decrease in the adoption rate (as opposed to the big increase in the original claim), exposing the actual trend after Giuliani took office.

**Marshall's Vote Correlation Claim** As another proof of concept, consider Marshall's vote correlation claim in Example 2.

Suppose we have the reverse-engineered claim as specified in Section 5.1. We ran the CA-po algorithm for TSS-CA$_{ab}$ to find counterarguments with other sensible comparison periods that yield lower vote correlations between Marshall and Boehner. The top counterarguments, in decreasing sensibility, perturb the start of the period to the beginning of 2009, 2008, and 2007, yielding decreasing correlations of $59.65\%$, $57.36\%$, and $53.63\%$. These results include the counterargument found by `factcheck.org`, and suggest that the vote correlation between Marshall and Boehner had not always been high.

## 6.2 Efficiency and Scalability of Algorithms

We now turn to experiments comparing the performance of three classes of algorithms—Base (baseline), Enum (enumeration-based), and DiCo (divide-and-conquer). For brevity, when the context is clear, we use these names to refer to the respective algorithms for a given problem. Because of limited space, we only present a sample of our results here. We focus mainly on finding counterarguments (CA), since algorithms for reverse-engineering (RE) are similar to CA, and the comparison among the three classes of algorithms also shows similar trends. We also focus more on WAC than on TSS. For the complete set of results, see [28].

Each data point in the figures below is obtained by averaging over 100 original claims with randomly generated parameter settings. For the results below, all algorithms (including Base) implement the preprocessing optimization (Sections 4.2.1 and 5.2.1).

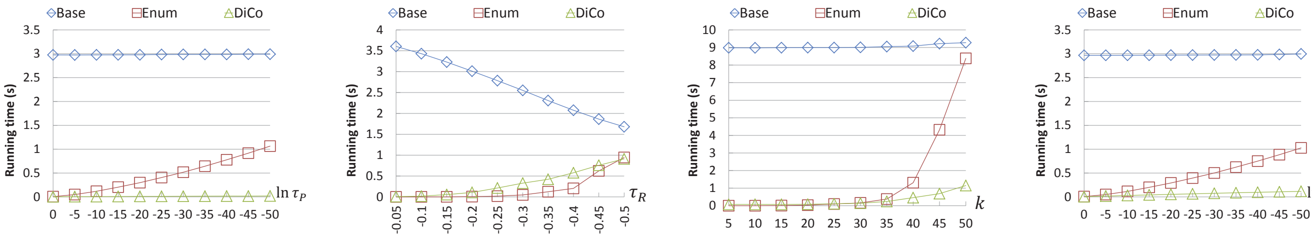
**Varying $\tau_{\mathcal{P}}$ in CA-$\tau_{\mathcal{P}}$ for WAC on UNEMP** Figure 6a shows the running times of the CA-$\tau_{\mathcal{P}}$ algorithms for WAC claims on UNEMP, as we vary the parameter sensibility threshold $\tau_{\mathcal{P}}$. Since Base always explores the entire parameter space $\mathcal{P}$, overall it is much slower than Enum and DiCo. However, as $\tau_{\mathcal{P}}$ decreases, the region of $\mathcal{P}$ meeting this threshold becomes larger. Since the radius of this region is $O(|\ln \tau_{\mathcal{P}}|^{1/2})$, Enum needs to explore $O(|\ln \tau_{\mathcal{P}}|^{3/2})$ settings, which explains Enum's super-linear increase in running time in Figure 6a. DiCo, with its powerful low-level building blocks, runs in time linear in $|\ln \tau_{\mathcal{P}}|$. This trend is difficult to see in the figure, as DiCo remains fast even with very low sensibility thresholds.

**Varying $\tau_{\mathcal{R}}$ in CA-$\tau_{\mathcal{R}}$ for WAC on UNEMP** Figure 6b considers the CA-$\tau_{\mathcal{R}}$ problem for WAC claims on UNEMP, and compares the algorithms as we vary the result strength threshold $\tau_{\mathcal{R}}$. Here, as $\tau_{\mathcal{R}}$ decreases, we want counterarguments with results that deviate farther from the original claim, which are harder for Enum and DiCo to find. On the other hand, lower $\tau_{\mathcal{R}}$ makes Base faster

[9] `http://data.bls.gov/timeseries/LNS14000000`

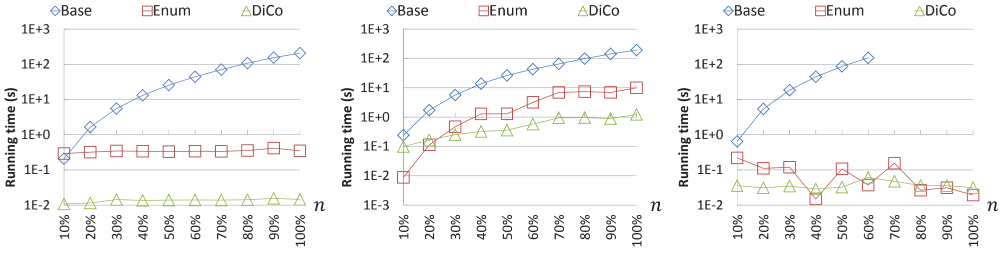
[10] `http://www.dot.state.tx.us/txdot_library/drivers_vehicles/publications/crash_statistics/`

[11] `http://www.govtrack.us/`



(a) CA-$\tau_{\mathcal{P}}$, varying $\tau_{\mathcal{P}}$ (b) CA-$\tau_{\mathcal{R}}$, varying $\tau_{\mathcal{R}}$ (c) CA-po, varying $k$ (d) RE-$\tau_{\mathcal{P}}$, varying $\tau_{\mathcal{P}}$

Figure 6: Running time of CA and RE algorithms for WAC claims on UNEMP.



(a) CA-$\tau_{\mathcal{P}}$ ($\ln \tau_{\mathcal{P}} = -20$) (b) CA-$\tau_{\mathcal{R}}$ ($\tau_{\mathcal{R}} = -0.05$) (c) CA-po ($k = 20$)

Figure 7: Running time of CA algorithms for WAC claims on AUTO when varying data size.

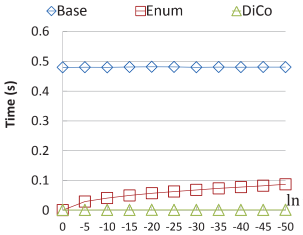


Figure 8: Running time of CA-$\tau_{\mathcal{P}}$ algorithms for TSS-CA$_{ab}$ on VOTE, varying $\tau_{\mathcal{P}}$.

because it needs to call SP with fewer parameter settings that meet the result strength threshold.[12] When $\tau_{\mathcal{R}}$ is as small as $-0.5$, a large portion of $\mathcal{P}$ must be examined by Enum and DiCo. In fact, counterarguments found at this point are starting to be no longer useful, because their parameter settings are already too far from the original claim. Thus, for practical values of $\tau_{\mathcal{R}}$, Enum and DiCo are faster than Base. Also, we see that for CA-$\tau_{\mathcal{R}}$, DiCo holds no advantage over Enum, which can be explained by the overhead of DiCo's exponential search in this case.

**Varying $k$ in CA-po for WAC on UNEMP** We now turn to CA-po, which returns the $k$ Pareto-optimal counterarguments with highest sensibility. As explained in Section 2.2, this problem formulation is attractive because it avoids the sometimes tricky task of choosing thresholds for CA-$\tau_{\mathcal{P}}$ and CA-$\tau_{\mathcal{R}}$. Figure 6c shows the running time of the three algorithms for WAC claims on UNEMP when we vary $k$. Enum and DiCo show comparable performance up to $k = 35$. After that, the running time of Enum increases rapidly, and approaches that of Base. On the other hand, the running time of DiCo shows a much slower increase and remains much faster than Base for all $k$ values tested.

**Varying $\tau_{\mathcal{P}}$ in RE-$\tau_{\mathcal{P}}$ for WAC on UNEMP** As a sample of reverse-engineering experiments, Figure 6d compares the three algorithms for RE-$\tau_{\mathcal{P}}$ for WAC claims on UNEMP. As we decrease the parameter sensibility threshold $\tau_{\mathcal{P}}$, we observe the same trend as in Figure 6a for CA-$\tau_{\mathcal{P}}$: Base is the slowest, while DiCo scales better than Enum in the size of the high-sensibility region of the parameter space. Note that DiCo for RE-$\tau_{\mathcal{P}}$ in Figure 6d is slightly slower than DiCo for CA-$\tau_{\mathcal{P}}$ in Figure 6a, because of the more expensive building block (OptP$_0$ vs. OptP$_{-\infty}$ in Section 4.2.3).

**Varying Data Size in CA for WAC on AUTO** Besides testing the performance of the algorithms while varying their input parameters, we also show how they scale with respect to data size. In Figure 7, we show the results on the three variants of the problem of finding counterarguments—CA-$\tau_{\mathcal{P}}$, CA-$\tau_{\mathcal{R}}$, and CA-po—as we change the data size by taking prefixes of the AUTO time series with varying lengths (from 10% to 100% of the whole series). For CA-$\tau_{\mathcal{R}}$ (Figure 7b), Enum shows a rate of increase in running time similar to Base, while DiCo shows a slower rate of increase. This increasing trend is expected because more data points lead to more counterarguments with required strength threshold. For CA-$\tau_{\mathcal{P}}$ (Figure 7a) and CA-po (Figure 7c), Base continues to suffer from bigger data sizes, but Enum and DiCo remains fast. The reason is that Enum and DiCo limit their search within high-sensibility neighborhoods around the original claims; a bigger dataset spanning a longer time period does not necessarily increase the size of these neighborhoods. For all three variant problems of CA, Enum and DiCo are orders of magnitude faster than Base.

**Varying $\tau_{\mathcal{P}}$ in CA-$\tau_{\mathcal{P}}$ for TSS-CA$_{ab}$ on VOTE** As a sample of experiments on TSS claims, we now turn to VOTE data. Figure 8 compares the three algorithms for TSS-CA$_{ab}$, i.e., fixing two voters and perturbing the time period $[a, b]$ to find counterarguments that show lower vote correlation than originally claimed. Here, we consider the CA-$\tau_{\mathcal{P}}$ variant of the problem, and decrease the parameter sensibility threshold $\tau_{\mathcal{P}}$ (thereby enlarging the region of $\mathcal{P}$ meeting this threshold). We observe trends similar to those in Figures 6a and 6d for WAC claims: DiCo performs best, while Base is the slowest by a big margin. The only notable difference is that the parameter space is 3-d for WAC but only 2-d here. Hence, Enum and DiCo fare better here with an increasing search space.

## 7 Related Work

A large body of work on uncertain data management [9, 1, 17] considers the effect of data perturbations on query results. Our study of query parameter perturbations offers a conceptual counterpoint—while uncertain databases consider one query over many database instances (possible worlds), we are interested in many queries (perturbed versions of each other) over one database instance. Interestingly, one could, at least in theory, mimic query parameter perturbations by constructing tables of uncertain query parameters, and "joining" them with data tables in some fashion to compute the QRS. However, the resulting query will be awkward and difficult to optimize. Furthermore, we are interested in certain questions about QRS—beyond computing a representation of the surface or computing expectations from it—that have not been the goal

[12] One might wonder why fewer SP calls matter so much. It turns out that in this case, thanks to precomputed prefix-sums, SR is much faster than SP, so the cost of SP calls dominates. This effect also explains why Base did not see visible improvement with fewer SR calls in Figure6a. In practice, when query evaluation is more expensive, the opposite may hold.

of query evaluation in uncertain databases. Nonetheless, uncertain data management offers many ideas relevant to fact-checking. For example, our future work (further discussed in Section 8) includes investigating sampling and approximation, and extending our model to consider parameter and data perturbations jointly.

The notion of response surfaces has appeared in various contexts, but usually with specific uses different from ours. In *parametric query optimization* [16, 11, 15, 8], a response surface represents the best query execution plan (and its cost) over the space of parameters relevant to query optimization, including system parameters, selectivity factors, and/or query parameters. In our recent work on publish/subscribe, we use QRS to succinctly represent (and index) answers to a large number of continuous linear preference top-$k$ queries with different parameter settings [29]. Similar ideas have been used in understanding the sensitivity of such top-$k$ rankings with respect to their parameter settings [23, 21]. Lin et al. [19] uses a surface to concisely represent how the set of association rules varies with support and confidence settings.

The reverse-engineering problem is related to recent work on *query by output* [26] and *view synthesis* [22], which tries to find queries returning a given result. Unlike these problems, we are given not only the claim result but also additional information—the context of the original claim (including any explicitly mentioned parameter values) and the query template. Tran and Chan [25] consider how to modify a query such that it returns both the original result as well as additional desired tuples. He and Lo [14] tackle the specific setting of linear preference top-$k$ queries. Wu and Madden [27] study how to use queries to explain away outliers in aggregate results. While the work discussed above is similar to our problem in spirit, their search spaces and goals are very different, and none of them models query perturbations probabilistically.

The idea of ordered enumeration, used by one of the meta algorithms discussed in this paper, has been applied in many settings. In particular, the problem of enumerating multidimensional vectors under a scoring function has been studied by Luo et al. [20] in the context of supporting keyword searches, and by Agrawal and Widom [2] in the context of joining two uncertain relations.

## 8 Conclusion and Future Work

In this paper, we have shown how to turn fact-checking into a computational problem. Interestingly, by regarding claims as queries with parameters, we can check them—not just for correctness, but more importantly, for more subtle measures of quality—by perturbing their parameters. This observation leads us to a powerful framework for modeling and for developing efficient algorithms for fact-checking tasks, such as reverse-engineering vague claims and countering questionable claims. We have shown how to handle real-world claims in our framework, and how to obtain efficient algorithms by supplying appropriate building blocks.

Our proposed framework has opened up more research problems than we can possibly hope to address in a single paper. There are several lines of work underway, including efficient computation of quality measures, approximation algorithms, unified modeling of parameter and data changes, and going from fact-checking to lead-finding. Specialized building blocks for many other claim types remain to be discovered. Besides the interesting problems at the back-end, we are also working on making our techniques easier to apply. Along this line, we are investigating a learning-based approach that rely on user feedback to help specify functions SR and SP. The culmination of this work will be an end-to-end system to empower journalists and the public in combating the "lies, d—ed lies, and statistics" that permeate our public life today. To that end, we also need advances in complementary research areas such as source identification, data integration, data cleansing, natural language querying, and crowdsourcing.

## References

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

[2] P. Agrawal and J. Widom. Confidence-aware join algorithms. *ICDE*, 2009, 628–639.

[3] A. M. Andrew. Another efficient algorithm for convex hulls in two dimensions. *Information Processing Letters*, 9(1979), 216–219.

[4] M. A. Bender and M. Farach-Colton. The LCA problem revisited. *LATIN*, 2000, 88–94.

[5] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. *ICDE*, 2001, 421–430.

[6] S Cohen, J. T. Hamilton, and F. Turner. Computational journalism. *CACM*, 54(2011), 66–71.

[7] S. Cohen, C. Li, J. Yang, and C. Yu. Computational journalism: A call to arms to database researchers. *CIDR*, 2011.

[8] Harish D., P. N. Darera, and J. R. Haritsa. Identifying robust plans through plan diagram reduction. *VLDB*, 2008, 1124–1140.

[9] N. N. Dalvi, C. Ré, and D. Suciu. Probabilistic databases: Diamonds in the dirt. *CACM*, 52(2009), 86–94.

[10] J. Fischer and V. Heun. A new succinct representation of rmq-information and improvements in the enhanced suffix array. *ESCAPE*, 2007, 459–470.

[11] S. Ganguly. Design and analysis of parametric query optimization algorithms. *VLDB*, 1998, 228–238.

[12] J. Gray, A. Bosworth, A. Layman, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-total. *ICDE*, 1996, 152–159.

[13] D. Harel and R. E. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM*, 13(1984), 338–355.

[14] Z. He and E. Lo. Answering why-not questions on top-k queries. *ICDE*, 2012, 750–761.

[15] A. Hulgeri and S. Sudarshan. AniPQO: Almost non-intrusive parametric query optimization for nonlinear cost functions. *VLDB*, 2003, 766–777.

[16] Y. E. Ioannidis, R. T. Ng, K. Shim, and T. K. Sellis. Parametric query optimization. *VLDB*, 1992, 103–114.

[17] R. Jampani, F. Xu, M. Wu, L. L. Perez, C. Jermaine, and P. J. Haas. The Monte Carlo database system: Stochastic analysis close to the data. *TODS*, 36(2011), 18.

[18] H. T. Kung, F. Luccio, and F. P. Preparata. On finding the maxima of a set of vectors. *JACM*, 22(1975), 469–476.

[19] X Lin, A. Mukherji, E. A. Rundensteiner, C. Ruiz, and M. O. Ward. PARAS: A parameter space framework for online association mining. *VLDB* 6(2013), 193–204.

[20] Y. Luo, X. Lin, W. Wang, and X. Zhou. Spark: top-k keyword query in relational databases. *SIGMOD*, 2007, 115–126.

[21] K. Mouratidis and H. Pang. Computing immutable regions for subspace top-k queries. *VLDB*, 6(2012), 73–84.

[22] A. Das Sarma, A. G. Parameswaran, H. Garcia-Molina, and J. Widom. Synthesizing view definitions from data. *ICDT*, 2010, 89–103.

[23] M. A. Soliman, I. F. Ilyas, D. Martinenghi, and M. Tagliasacchi. Ranking with uncertain scoring functions: Semantics and sensitivity measures. *SIGMOD*, 2011, 805–816.

[24] R. E. Tarjan. Applications of path compression on balanced trees. *JACM*, 26(1979), 690–715.

[25] Q. T. Tran and C. Y. Chan. How to ConQueR why-not questions. *SIGMOD*, 2010, 15–26.

[26] Q. T. Tran, C. Y. Chan, and S. Parthasarathy. Query by output. *SIGMOD*, 2009, 535–548.

[27] E. Wu and S. Madden. Scorpion: Explaining away outliers in aggregate queries. *VLDB*, 6(2013), 553–564.

[28] Y. Wu, P. K. Agarwal, C. Li, J. Yang, and C. Yu. Toward computational fact-checking. Technical report, Duke University, 2013. `http://www.cs.duke.edu/dbgroup/papers/WuAgarwalEtAl-13-fact_check.pdf`.

[29] A. Yu, P. K. Agarwal, and J. Yang. Processing a large number of continuous preference top-k queries. *SIGMOD*, 2012, 397–408.