Computational Fact Checking through Query Perturbations

YOU WU and PANKAJ K. AGARWAL, Duke University
CHENGKAI LI, University of Texas at Arlington
JUN YANG, Duke University
CONG YU, Google Research

Our media is 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, for example, is a claim "cherry-picking"? This article proposes a framework that models claims based on structured data as parameterized queries. Intuitively, with its choice of the parameter setting, a claim presents a particular (and potentially biased) view of the underlying data. A key insight is that we can learn a lot about a claim by "perturbing" its parameters and seeing how its conclusion changes. For example, a claim is not robust if small perturbations to its parameters can change its conclusions significantly. This framework allows us to formulate practical fact-checking tasks—reverse-engineering 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.

Categories and Subject Descriptors: H.2.8 [**Database Management**]: Database Applications

General Terms: Design, Algorithms

Additional Key Words and Phrases: Sensitivity analysis, fact checking, computational journalism

ACM Reference Format:
You Wu, Pankaj K. Agarwal, Chengkai Li, Jun Yang, and Cong Yu. 2017. Computational fact checking through query perturbations. ACM Trans. Database Syst. 42, 1, Article 4 (January 2017), 41 pages.
DOI: http://dx.doi.org/10.1145/2996453

1. INTRODUCTION

While much of 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

You Wu is now at Google Research.
This work is supported by NSF grants IIS-14-08846 and IIS-14-08928. P.A. is also supported by NSF grants CCF-11-61359, CCF-15-13816, and CCF-15-46392, by an ARO grant W911NF-15-1-0408, and by Grant 2012/229 from the U.S.-Israel Binational Science Foundation. C.L. is also supported by NSF grant IIP-15-65699 and a Knight Prototype Fund. J.Y. is also supported by NSF grant IIS-13-20357 and a Google Faculty Research Award.
Authors' addresses: Y. Wu and C. Yu, Structured Data Group, Google Research, 111 8th Avenue, New York, NY 10011, USA; emails: {wuyou, congyu}@google.com; P. K. Agarwal, Department of Computer Science, Levine Science Research Center D214A, Duke University, Box 90129, Durham, NC 27708-0129, USA; email: pankaj@cs.duke.edu; C. Li, Department of Computer Science and Engineering, Engineering Research Building (ERB) Room 628, The University of Texas at Arlington, 500 UTA Boulevard, Arlington, TX 76019-0015; email: cli@uta.edu; J. Yang, Department of Computer Science, Levine Science Research Center D327, Duke University, Box 90129, Durham, NC 27708-0129, USA; email: junyang@cs.duke.edu.

DOI: http://dx.doi.org/10.1145/2996453

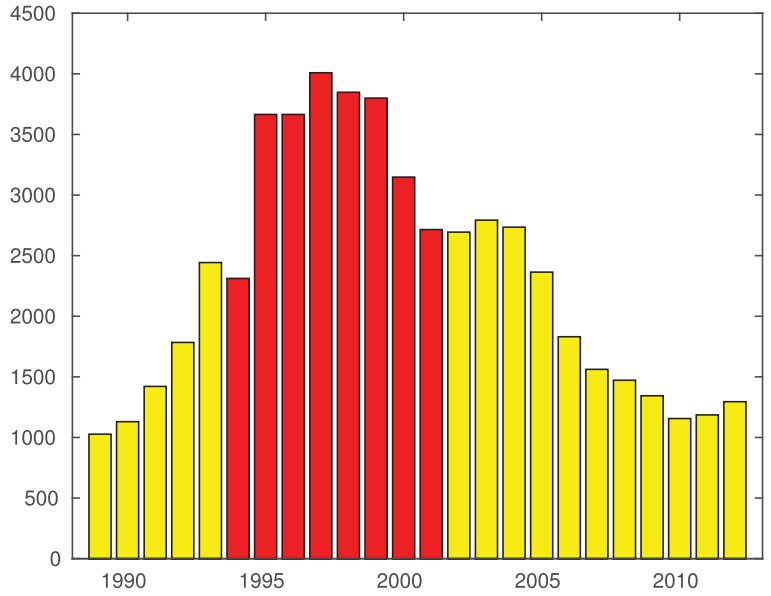


Fig. 1. New York City adoptions by year, 1989–2012. Giuliani's years in red.

asking queries that lead to high-quality claims. But first, what does "quality" mean? Consider the following.

Example 1.1 (*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 New York City "when he was the mayor." More precisely, the comparison was 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 did the claim compare these two particular 6-year periods? As it turns out (as shown in Figure 1), the underlying data reveal 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. However, if we compare Giuliani's first and second 4-year terms, that is, 1994–1997 and 1998–2001, we will see that the total number of adoptions in fact decreased by 1%.[1]

Example 1.2 (*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."[2] 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%, which is not very high considering the fact that even the Democratic Whip, Jim Clyburn, voted 44% of the

[1]The original factcheck.org article countered Giuliani's claim by comparing the adoption rates at the beginning and the end of his tenure, which led to a smaller increase of 17%. Here, we use a different counterargument found by our proposed method, because comparing 1-year windows is more susceptible to fluctuations in data.

[2]This ad was in response to an earlier ad attacking Marshall's "party loyalty votes" for voting with Nancy Pelosi (a notable Democrat as the Speaker of the U.S. House of Representatives at the time of the claim) "almost 90 percent of the time," which, not surprisingly, also tailored the claim in ways to further its own argument, by quoting a high voting correlation with the Republican leadership.

time with Boehner during that period. Basically, many votes in Congress are not as controversial as the public would think!

For both of the previously mentioned claims, we can verify their correctness using SQL queries over reliable, structured datasets available to the public. Database systems are good at verifying whether these claims are correct, but from the preceding discussion, it is obvious that assessing claim quality involves much more than testing correctness. Indeed, both of the previously mentioned claims are correct 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 previously mentioned claims 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. Can we automate this process of "fact checking," making it easier and more effective?

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 1.1 and 1.2)? 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 nonexpert public audience, understand why a claim has poor quality. As examples, we counter Giuliani's adoption claim with the argument that "*. . . compare the Giuliani's first and second 4-year terms, . . . the total number of adoptions in fact decreased by 1%*"; 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 1.1 and 1.2 omit important details such as the exact time periods of comparison. Numbers are usually 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 previously mentioned challenges, we propose in Section 2 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, in Section 2.1, 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 *Query Response Surface (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 in Section 2.2 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 also in Section 2.2 how to formulate fact-checking tasks—such as *finding counterexamples* 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 1.1) in

Table I. Notations for the Modeling Framework

Notation	Description
$\mathcal{P}$	parameter space
p	parameter setting
p_0	parameter setting of the original claim
$\mathcal{R}$	result space
r	result of a claim
r_0	result of the original claim
$q : \mathcal{P} \to \mathcal{R}$	parameterized query template
$\mathsf{SP}(p; p_0)$	sensibility of parameter p relative to p_0
$\mathsf{SR}(r; r_0)$	strength of result r relative to r_0

Section 4.1 and *time series similarity claims* (generalization of the Marshall-Boehner vote correlation claim in Example 1.2) in Section 5.1.

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. It is not surprising that more efficient algorithms are only possible by exploiting claim-specific properties, but to make our techniques broadly applicable, we want to develop more generic algorithms that require little claim-specific configuration:

—We propose "meta" algorithms in Section 3 that work across different claims that share the same (general) properties, or those for which certain low-level algorithmic building blocks are available.
—For window aggregate comparison claims and time series similarity claims, we develop efficient, specialized algorithmic building blocks in Sections 4.2 and 5.2, respectively, that can be plugged into the preceding meta algorithms to enable fact checking in real time for interactive users.

Our approach enables an extensible system architecture with "pay-as-you-go" support for efficient fact checking—more efficient support can be enabled without reimplementing the high-level meta algorithms, but simply by plugging in instantiations of low-level building blocks.

Finally, we experimentally validate the ability of our framework in modeling fact-checking tasks and producing meaningful results in Section 6.1, as well as the efficiency of our computational techniques in Section 6.2.

We herein focus on the challenges of finding counterexamples and reverse-engineering vague claims. We also 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. We briefly discuss these other challenges in Section 7.

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 setting 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. Notations are summarized in Table I.

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.[3] The QRS of q is the "surface" $\{(p, q(p)) \mid p \in \mathcal{P}\}$ in $\mathcal{P} \times \mathcal{R}$.[4]

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 in its neighborhood.

For example, to check Giuliani's claim in Example 1.1, suppose we have a table adopt(year, number) of yearly adoption numbers. 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 $\langle \texttt{Q1}, (w = 6, t = 2001, d = 6), 1.665\rangle$.

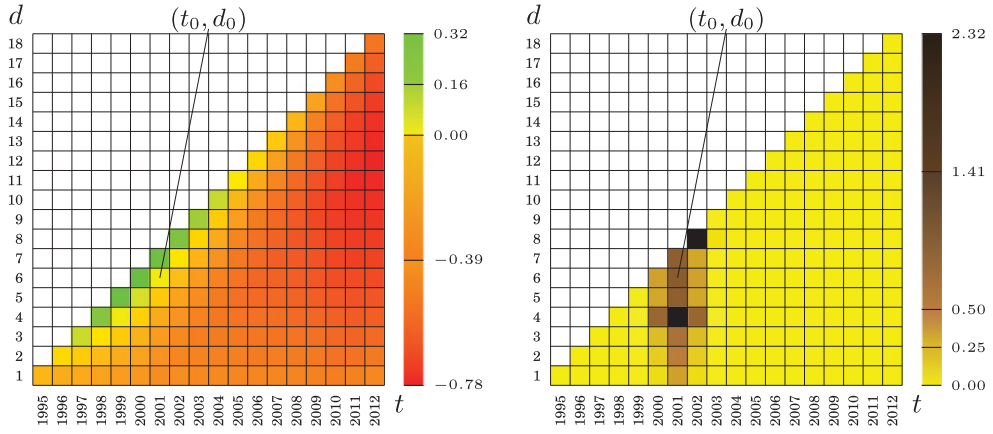
Relative Strength of Results. To capture the effect of parameter perturbations on query results, we need a way to compare results. For example, if a perturbation in Giuliani's claim leads to a lower increase (or even decrease) in the total adoption number, this new result is "weaker" than the result 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, respectively) 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 $\mathcal{P} \times \mathbb{R}$. We call this simplified surface the *relative result strength surface*. For example, Figure 2(a) illustrates this surface for Giuliani's adoption claim. Since a surface in $\mathbb{R}^4$ is difficult to visualize, we fix w to 6 and plot SR 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

[3]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.

[4]Strictly speaking, for this set to be regarded a surface, we need $\mathcal{P} \times \mathcal{R}$ to be continuous. In general, $\mathcal{P}$ can be discrete or even categorical, and $\mathcal{R}$ is the powerset of all possible result tuples. In this case, we assume that $\mathcal{P}$ is sampled from some underlying continuous space and the query results can be mapped to a numeric domain and are interpolated between sampled points to form a surface.



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

Fig. 2. 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 constraint $t - d - w \geq 1988$; 1989 is when the data became available.

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 an intrinsic property of the domain, "relevance" is 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 2(b) 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. Here, we model relevance using a Gaussian kernel centered at p_0, because intuitively, parameter settings far away from p_0 are not relevant to the claim context. We model naturalness based on the observation that it is more natural to compare periods that are 4 or 8 years apart, because New York City mayors have 4-year terms. The combination (multiplication) of naturalness and relevancy results in generally decaying sensibility around $(t_0, d_0) = (2001, 6)$ (because of relevancy), but with bumps when $d = 4$ and $d = 8$ (because of naturalness). 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 clear choice of 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 the parameter space $\mathcal{P}$: $p_1 \preceq^{p_0} p_2$ 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 Section 5 for more details).

Assumption on Data. We assume that the dataset is "sufficient" for checking the claim, in the sense that a sensibility function appropriately defined by a domain expert would not consider perturbing parameter settings beyond what the underlying dataset can support (e.g., if a perturbation requires accessing historical data that was not recorded).

Relationship between Query Perturbations and Data. While query perturbations may appear data agnostic, they are in fact governed by domain knowledge, through the specification of parameter sensibility function. Such domain knowledge can be reflected by data, and may even be derived from data. However, there are many cases where exploration is better guided by domain knowledge that cannot be derived by data alone. For example, the yearly adoption dataset by itself cannot tell us that comparing across the terms of a mayor is more natural.

At the same time, it is possible to exploit data characteristics for more efficient processing. For example, the sparsity in data may readily imply that results for some nearby parameter settings are identical. As we shall see in Sections 4 and 5, for both Window Aggregate Comparison (WAC) and Time Series Similarity (TSS), we use this idea in our algorithms by employing prefix sums computed over the data, which effectively identify the locations of possible result changes in the parameter space, without having to consider every parameter setting.

2.2. Formulating Fact-Checking Tasks

We now show how to cast various fact-checking tasks as questions about the QRS with the help of our framework.

Finding Counterarguments. Given original claim $\langle q, p_0, r_0\rangle$, a *counterargument* is a parameter setting p such that $\mathsf{SR}(q(p); r_0) < 0$; that is, it weakens the original claim. For example, Figure 2(a) 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 $\preceq^{p_0}$; that is, 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 in descending order of parameter setting sensibility, until the desired counterargument is spotted.[5] This problem is formulated below:

[5]It is easy to see that when SP exists, for any $\tau_{\mathcal{R}}$ ($\tau_{\mathcal{P}}$), CA-$\tau_{\mathcal{R}}$(CA-$\tau_{\mathcal{P}}$) always finds a Pareto-optimal counterargument. The same is true for RE-$\tau_{\mathcal{R}}$ and RE-$\tau_{\mathcal{P}}$ problems.

(CA-po). This problem requires the parameter sensibility function SP. Given original claim $\langle q, p_0, r_0\rangle$ and an integer $k > 0$, find the k Pareto-optimal counterarguments $p \in \mathcal{P}$ with the highest $\mathsf{SP}(p; p_0)$ values. 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 preceding problems. A *Pareto-optimal* counterargument is one that is dominated by no counterarguments.

Reverse-Engineering Vague Claims. As motivated in Section 1, many claims are not stated precisely or completely; for example, 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, that is, $(w_0, t_0, d_0) = (1, 2001, 8)$, to capture the claim context that Giuliani was in office 1994–2001 (p_0 represents the comparison between two 1-year windows—the year before Giuliani's term and the last year of his term). Note that when picking p_0, we do not need to tweak it to make $q(p_0)$ match r_0; with our problem formulation below, this reverse-engineering task will be carried out automatically.

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, analogous to those for finding counterarguments.

(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 $\preceq^{p_0}$.

(RE-$\tau_{\mathcal{P}}$). Find, given vague 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.

(RE-po). Given vague claim $\langle q, p_0, r_0\rangle$, and an integer $k > 0$, find the k Pareto-optimal reverse-engineered parameter settings $p \in \mathcal{P}$ with the highest $\mathsf{SP}(p; p_0)$ values. We say that a reverse-engineered parameter setting *pdominates* another one p' if (i) $p \succeq^{p_0} p$; (ii) $|\mathsf{SR}(q(p); r_0)| \leq |\mathsf{SR}(q(p'); r_0)|$; and (iii) inequality is strict for at least one of the preceding variants. A *Pareto-optimal* reverse-engineered parameter setting is one that is dominated by no others.

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),[6] or, if $\mathcal{P}$ is continuous, a probability density function (pdf). Consider a "random fact-checker," who randomly perturbs the parameter setting according to 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.

There are a number of meaningful measures of claim quality; we highlight three here. For simplicity of exposition, we assume that $\mathcal{P}$ is finite and discrete, and that SP is a pmf. Generalization to the continuous case is straightforward.

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

(Fairness). The *fairness* of a claim $\langle q, p_0, r_0 \rangle$ is

$$\sum_{p \in \mathcal{P}} \mathsf{SP}(p; p_0) \cdot \mathsf{SR}(q(p); r_0). \tag{1}$$

Intuitively, fairness is the expected strength (relative to r_0) of a perturbed claim generated by the random fact-checker. Fairness of 0 means the claim is unbiased; positive fairness means the claim is understated; negative fairness means it is overstated.

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

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

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.

(Uniqueness). The *uniqueness* [Wu et al. 2012] of a claim $\langle q, p_0, r_0 \rangle$ is

$$\frac{1}{|\mathcal{P}|} \cdot \sum_{p \in \mathcal{P}} \mathbf{1}\big(\mathsf{SR}(q(p); r_0) < 0\big). \tag{3}$$

Here, $\mathbf{1}(\cdot)$ is an indicator function. In other words, uniqueness is the fraction of all possible parameter settings that yield results weaker than the original claim. This definition does not require a parameter sensibility relation or function (or it can be seen as assuming a uniform pmf $\mathsf{SP}(p; p_0) = \frac{1}{|\mathcal{P}|}$). Low uniqueness means it is easy to find perturbed claims that are at least as strong as the original one.

Different quality measures make sense for claims of different types, or the same claim viewed from different perspectives. For example, for a claim that singles out some "entity" to be special—for example, the Marshall-Boehner vote correlation claim in Example 1.2, or *one-of-the-few* claims in Wu et al. [2012]—uniqueness measures how special this entity really is among its peers. In this case, a peak in QRS (over perturbations to the entity) is rewarded. On the other hand, for a claim whose context is set by a condition—for example, the Marshall-Boehner claim compute vote correlation over a particular time period—fairness and robustness measure how the claim holds up in different contexts. In this case, a peak in QRS (over perturbations to the condition) is penalized.

Also, note that a claim can be fair but not robust—in that case, the claim reflects the "average" case but still can be easily weakened because of variability in the QRS. For example, Giuliani's claim fares better in fairness than in robustness. For fairness, those perturbations that strengthen this claim (recall Figure 2) can somewhat offset the perturbations that weaken it. For robustness, however, the strengths of the stronger claims do not contribute to the measure, which focuses on the weaker claims.

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 want to develop more generic algorithms than those specialized for particular types of claims. However, more efficient algorithms are only possible by exploiting specific properties of the claims. To

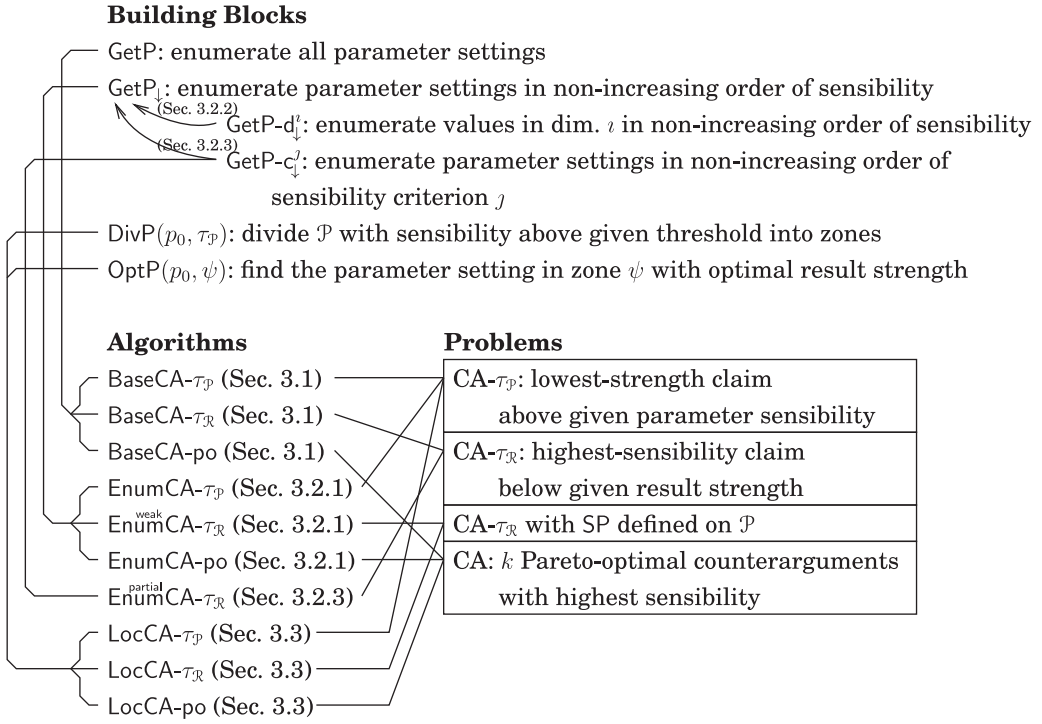


Fig. 3. Notations for the algorithmic framework.

gain efficiency without sacrificing generality, we develop a series of "meta" algorithms (also summarized in Figure 3):

—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 locus approach for searching the parameter space (Section 3.3). Instantiations of such building blocks for claims generalizing Examples 1.1 and 1.2 will be presented in Sections 4.2 and 5.2, respectively.
—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 present examples of how plugged-in preprocessing helps with checking claims in Examples 1.1 and 1.2.

This approach enables an extensible system architecture with "pay-as-you-go" support for efficient fact checking—new claim types can be supported with little configuration effort up front, but more efficient support can be enabled by plugging in instantiations of low-level building blocks, without re-implementing the high-level meta algorithms.

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 $\mathsf{SR}(q(p); r_0)$, these algorithms use $|\mathsf{SR}(q(p); r_0)|$ instead; other aspects remain the same.

When analyzing the time complexity of the meta algorithms below, we use μ_q to denote the cost (in time) of evaluating the query template with a specific parameter setting; we use μ_p to denote the cost of one GetP call. For brevity, we assume that the cost of computing $\mathsf{SR}(q(p); r_0)$ is dominated by that of computing $q(p)$, so it is $O(\mu_q)$; we assume the cost of $\mathsf{SP}(p; p_0)$ is $O(\mu_p)$. The space complexity of the meta algorithms below does not include the space used for evaluating queries or calling the plugin functions—unless otherwise noted, the total space consumption is bounded by their sum, and is not affected by the number of times that these queries and functions are evaluated.

3.1. Baseline Algorithms

The baseline algorithms assume nothing beyond the minimum required to define the problems. To find counterarguments, these algorithms simply call GetP to consider all possible parameter settings exhaustively. More details are presented below. Suppose the original claim is $\langle q, p_0, r_0 \rangle$.

(BaseCA-$\tau_{\mathcal{P}}$). Given a parameter sensibility threshold $\tau_{\mathcal{P}}$, BaseCA-$\tau_{\mathcal{P}}$ solves CA-$\tau_{\mathcal{P}}$ as follows. For each parameter setting p obtained from GetP, if p's sensibility $\mathsf{SP}(p; p_0) > \tau_{\mathcal{P}}$, and if its result strength $\mathsf{SR}(q(p); r_0)$ is no less than the lowest result strength seen so far, BaseCA-$\tau_{\mathcal{P}}$ remembers p and its result strength. After considering all $p \in \mathcal{P}$, BaseCA-$\tau_{\mathcal{P}}$ returns its remembered parameter setting(s).

Base CA-$\tau_{\mathcal{P}}$ makes $|\mathcal{P}|$ calls to GetP, SP, q, and SR; therefore, its time complexity is $\Theta(|\mathcal{P}|(\mu_p + \mu_q))$. BaseCA-$\tau_{\mathcal{P}}$ may need to remember multiple settings that tie for the lowest result strength so far. Such ties are rare in practice, so BaseCA-$\tau_{\mathcal{P}}$ usually uses constant space.

(BaseCA-$\tau_{\mathcal{R}}$). Given a result strength threshold $\tau_{\mathcal{R}}$, BaseCA-$\tau_{\mathcal{R}}$ solves CA-$\tau_{\mathcal{R}}$ by calling GetP and considering each parameter setting p with $\mathsf{SR}(q(p); r_0) < \tau_{\mathcal{R}}$—for brevity, we call such parameter settings *qualified*. BaseCA-$\tau_{\mathcal{R}}$ remembers the maximal subset A of all qualified parameter settings it has seen so far. To update A, BaseCA-$\tau_{\mathcal{R}}$ compares p to each element $p' \in A$: it removes all p' from A where $p' \preceq^{p_0} p$, and adds p to A if $p \not\preceq^{p_0} p'$ for all $p' \in A$. After considering all $p \in \mathcal{P}$, BaseCA-$\tau_{\mathcal{R}}$ returns A.

Let $t \leq |\mathcal{P}|$ denote the maximum size reached by A during execution. BaseCA-$\tau_{\mathcal{R}}$ makes $|\mathcal{P}|$ calls to GetP, q, and SR, and $O(|\mathcal{P}|t)$ calls to $\preceq^{p_0}$. Its time complexity is $O(|\mathcal{P}|(t\mu_p + \mu_q))$ and its space complexity is $O(t)$.

Note that BaseCA-$\tau_{\mathcal{R}}$ gracefully handles the special (and common) case where $\preceq^{p_0}$ defines a weak order (e.g., when $\preceq^{p_0}$ is derived from $\mathsf{SP}(p; p_0)$). In this case, A contains only the qualified parameter settings that tie for the highest sensibility, and the total number of calls to $\preceq^{p_0}$ is only $|\mathcal{P}|$.

(BaseCA-po). 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 D (sensibility and strength) [Kung et al. 1975; Börzsönyi et al. 2001]. 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 $t \leq |\mathcal{P}|$ 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.

Base CA-po makes $|\mathcal{P}|$ calls to GetP, SP, q, and SR. It runs in $O(|\mathcal{P}|(\mu_p \log t + \mu_q)$ time with $O(t)$ space.

The baseline algorithms are practical for small parameter spaces. Their running times, however, 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. Next, we propose more efficient algorithms enabled by additional knowledge of the problem instances.

3.2. Ordered Enumeration of Parameters

We now develop algorithms that take advantage of functions that enumerate, on demand, all possible parameter settings in nonincreasing order of sensibility. Ties are broken arbitrarily. When the parameter space $\mathcal{P}$ is large, such functions enable us to focus first on exploring the most sensible parts of $\mathcal{P}$. 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 GetP, which we denote by *GetP*$_\downarrow$, is available for generating parameter settings with high SP first.

The second case (Section 3.2.2) extends the first, and is rather common for multidimensional parameter spaces. Here, instead of requiring GetP$_\downarrow$, we require, for each axis $\imath$, a function *GetP-d*$^\imath_\downarrow$ for enumerating the values along this axis in nonincreasing order. We assume this order is consistent with the overall sensibility: that is, given a parameter setting p, replacing its value for axis $\imath$ 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 GetP-d$^\imath_\downarrow$'s to implement a single multidimensional GetP$_\downarrow$, so we can then apply the algorithms for the first case mentioned previously. 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 axes for w, t, and d individually, but we do not need to define how to enumerate (w, t, d) settings manually.

In the third case (Section 3.2.3), again, no single GetP$_\downarrow$ is given, but parameter settings can be compared according to multiple criteria. For each criterion $\jmath$, a function *GetP-c*$^\jmath_\downarrow$ exists to enumerate parameter settings in order according to $\jmath$. The GetP-c$^\jmath_\downarrow$'s together define a partial order $\preceq^{p_0}$ on $\mathcal{P}$. For example, as we will see in Section 5, the Marshall-Boehner vote correlation claim, as far as entity permutation is concerned, falls into this case—U.S. Representatives can be ordered by either recognizability or ideology, and the two criteria together define a partial order. Furthermore, if a parameter sensibility function SP can be defined by monotonically combining scores for these criteria, we provide a general procedure[7] that uses the multiple GetP-c$^\jmath_\downarrow$'s to implement a single overall GetP$_\downarrow$, so we can apply the algorithms for the first case.

We now present our algorithms for the previously mentioned cases. In the following, as an abuse of notation for simplicity, we shall continue using μ_p to denote the time for all GetP$_\downarrow$ variants, but keep in mind that μ_p varies across different instances of these functions.

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

(*EnumCA-*$\tau_\mathcal{P}$). Given a parameter sensibility threshold $\tau_\mathcal{P}$, EnumCA-$\tau_\mathcal{P}$ calls GetP$_\downarrow$ repeatedly until it gets a parameter setting p with $\mathsf{SP}(p; p_0) \leq \tau_\mathcal{P}$, at which

[7]Note that this procedure differs from the analogous procedure for the second, previously mentioned, case; see the end of Section 3.2.3 for detailed discussion.

point EnumCA-$\tau_{\mathcal{P}}$ terminates and returns the parameter settings with the lowest relative strength seen so far.

Enum CA-$\tau_{\mathcal{P}}$ improves on BaseCA-$\tau_{\mathcal{P}}$ by examining only those parameter settings whose sensibility is above the threshold.

(*EnumCA-$\tau_{\mathcal{R}}$*). EnumCA-$\tau_{\mathcal{R}}$ solves CA-$\tau_{\mathcal{R}}$ for the special case where a weak order on $\mathcal{P}$ is available (recall from Section 2.2 that CA-$\tau_{\mathcal{R}}$ requires only a partial order; that more general case will be discussed in Section 3.2.3). Given a result strength threshold $\tau_{\mathcal{R}}$, EnumCA-$\tau_{\mathcal{R}}$ calls $\mathsf{GetP}_{\downarrow}$ repeatedly until it gets a qualified parameter setting p, that is, with $\mathsf{SR}(q(p); r_0) < \tau_{\mathcal{R}}$. At this point, EnumCA-$\tau_{\mathcal{R}}$ knows that p is an answer because unseen parameter settings must have equal or lower sensibility. To get remaining answers (that tie with p in sensibility), EnumCA-$\tau_{\mathcal{R}}$ continues until $\mathsf{GetP}_{\downarrow}$ yields a parameter setting with lower sensibility.

Enum CA-$\tau_{\mathcal{R}}$ improves on BaseCA-$\tau_{\mathcal{R}}$ because EnumCA-$\tau_{\mathcal{R}}$ can terminate as soon as it finds an answer and finishes examining other parameter settings with the same sensibility. Also, its space complexity improves to $O(1)$ because there is no need to remember A as BaseCA-$\tau_{\mathcal{R}}$ does.

(*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 nonincreasing order of sensibility, 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 a 2 D skyline [Kung et al. 1975]. Furthermore, $|A|$ grows monotonically, so we can terminate once $|A|$ reaches k. For details, see Algorithm 1.

Enum CA-po runs in $O(s(\mu_p + \mu_q))$ time, 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. The improvement over BaseCA-po ($O(|\mathcal{P}|(\mu_p \log t + \mu_q))$ time and $O(t)$ space) is significant in practice, because oftentimes $s \ll |\mathcal{P}|$ and $k \ll t$.

Recall that we introduced CA-po in Section 2.2 as a method for interactive exploration when the choices of thresholds $\tau_{\mathcal{R}}$ and $\tau_{\mathcal{P}}$ are unclear for CA-$\tau_{\mathcal{R}}$ and CA-$\tau_{\mathcal{P}}$, respectively. With $\mathsf{GetP}_{\downarrow}$ and EnumCA-po, it is convenient to solve CA-po incrementally without a predetermined value for k, thereby enabling interactive exploration.

3.2.2. Enumerating Values along Each Axis. Given the parameter setting p_0 of the original claim, suppose that for each axis ι of the parameter space, a function $\mathsf{GetP\text{-}d}^{l}_{\downarrow}(p_0)$ is available for returning values for axis ι in order, such that $\mathsf{SP}(p; p_0)$ is monotonically nonincreasing with respect to the ordinal number of p's value for axis ι in the sequence returned by $\mathsf{GetP\text{-}d}^{l}_{\downarrow}(p_0)$. Note that it is possible for some combinations of single-dimensional values to be an invalid parameter setting. To comply with the assumption on data stated at the end of Section 2.1, 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 $\mathsf{GetP\text{-}d}^{l}_{\downarrow}$'s.

A simple strategy is to obtain the list of values for each axis ι by repeatedly calling $\mathsf{GetP\text{-}d}^{l}_{\downarrow}(p_0)$, perform the Cartesian product of all lists to generate all parameter settings, and then sort those that are valid. Suppose the number of values per axis is $O(\eta)$. The space complexity is $O(\eta^{\dim(\mathcal{P})})$ and $\Omega(|\mathcal{P}|)$. The time complexity is $O(\eta^{\dim(\mathcal{P})} + |\mathcal{P}|(\mu_q + \log|\mathcal{P}|))$.

We can do better by exploiting the monotonicity of SP and calling $\mathsf{GetP\text{-}d}^{l}_{\downarrow}(p_0)$ only when needed. The details are in Algorithm 2. We maintain the set of candidate parameter settings in a priority queue Q, whose priority is defined by sensibility. (Q.add(e, s) adds entry e with priority s to Q; Q.removeMax() deletes the entry e with the highest

ALGORITHM 1: EnumCA-po($\langle q, p_0, r_0 \rangle, k$).

```
1  A ← ∅; γ_A ← ∞;                                   // γ_A always tracks the strength for the last answer in A
2  p ← GetP↓(p_0);
3  while |A| < k do
4      Δ ← ∅;                                        // next set of answers, which tie with sensibility ρ and strength γ_Δ
5      ρ ← SP(p; p_0); γ_Δ ← ∞;
6      while SP(p; p_0) = ρ do                       // consider all parameter settings with sensibility ρ
7          γ ← SR(q(p); r_0);
8          if γ < γ_A then                           // to be an answer, strength must be lower than γ_A
9              if γ < γ_Δ then                       // a new low strength is found
10                 Δ ← {p}; γ_Δ ← γ;
11             else if γ = γ_Δ then                  // same strength; a tie
12                 Δ ← Δ ∪ {p};
13         p ← GetP↓(p_0);
14         if p = ⊥ then return A ∪ Δ;
15     if Δ ≠ ∅ then                                 // add new Pareto-optimal answers
16         A ← A ∪ Δ; γ_A ← γ_Δ;
17 return A;
```

priority s in Q, and returns (e, s); Q.contains(e) tests if Q contains entry e.) Q is initialized with an entry whose components are obtained by calling each GetP-d$^{\iota}_{\downarrow}(p_0)$ for the first time. Because of the monotonicity of SP, this entry has the highest sensibility in $\mathcal{P}$. We always remove the highest-priority entry from Q and, if it is valid, return it as the next parameter setting for GetP$_{\downarrow}$. Suppose the entry removed is $(v_1, \ldots, v_{\dim(\mathcal{P})})$. We insert the "successors" of this entry into Q as candidate parameter settings. Two entries e and e' form a predecessor-successor relationship if e' can be obtained from e by replacing one component v_ι with the value that GetP-d$^{\iota}_{\downarrow}(p_0)$ returns after v_ι. On the implementation end, for each axis ι, we use V_ι to cache the values returned by GetP-d$^{\iota}_{\downarrow}$ as a list. (V_ι.len() returns the length of the list; V_ι.get(j), where $0 \le j < V_\iota$.len(), returns the j-th value; V_ι.append(v) appends v to the list.) And for the priority queue Q, instead of storing the value for each axis ι, we store the index of this value in V_ι. Line 19 further ensures that we insert each candidate parameter setting into Q exactly once (when its first predecessor in lexicographical order is returned). For brevity of presentation, we assume there are no ties; it is straightforward to extend the results here to handle the general case involving ties.

The space complexity of this algorithm is $O(\eta^{\dim(\mathcal{P})-1})$, which is dominated by the priority queue;[8] cached values from GetP-d$^{\iota}_{\downarrow}$ 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 axes, 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 for a concrete problem, where we also show an example of constructing such GetP-d$^{\iota}_{\downarrow}(p_0)$ functions from the SP function (Algorithm 4).

[8]To see this bound, note that Q never contains two entries where one precedes the other (immediately or transitively) in the partial order induced by the single-dimensional orders. This property is ensured by the order of processing and line 19 of Algorithm 2. Therefore, the projection of entries in Q onto any subspace of dimensionality $\dim(\mathcal{P}) - 1$ is one-to-one.

ALGORITHM 2: $\mathsf{GetP}_{\downarrow}(p_0)$ using$\mathsf{GetP\text{-}d}^{\cdot}_{\downarrow}(p_0)$ and $\mathsf{IsPValid}(\cdot; p_0)$.

```
1  d ← dim(𝒫);
2  V_ι ← ∅ foreach ι ∈ [1, d];
3  def nextV_ι(j) begin
4  |  if j = V_ι.len() − 1 then
5  |  |  v ← GetP-d^ι_↓(p_0);
6  |  |  if v = ⊥ then return ⊥;
7  |  |  V_ι.append(v);
8  |  return j + 1;
9  Q ← ∅;
10 def p(⟨j_1, j_2, ..., j_d⟩) begin
11 |  return (V_1.get(j_1), V_2.get(j_2), ..., V_d.get(j_d));
12 Q.add(⟨nextV_1(−1), ..., nextV_d(−1)⟩, SP(p(⟨0, ..., 0⟩ (d zeros)); p_0));
13 while Q ≠ ∅ do
14 |  (e, s) ← Q.removeMax();                    // remove the highest-sensibility entry
15 |  for ι ← 1 to d do                          // add successor e' along each axis
16 |  |  e' ← e; e'[ι] ← nextV_ι(e[ι]);
17 |  |  if e'[ι] ≠ ⊥ then
18 |  |  |  Q.add(e', SP(p(e'); p_0));
19 |  |  if e[ι] > 0 then break;
20 |  if IsPValid(p(e); p_0) then yield p(e);
```

3.2.3. Enumerating Parameters by Criteria. Finally, consider the case where parameter settings can be compared according to multiple criteria. More formally, given the parameter setting p_0 of the original claim, suppose that for each criterion $_J$, $\mathsf{GetP\text{-}c}^{J}_{\downarrow}(p_0)$ is available for generating parameter settings in decreasing order according to $_J$ (again, we assume no ties for brevity; it is straightforward to extend our results to handle the general case). The parameter sensibility relation $p_1 \preceq^{p_0} p_2$ is defined as $\forall_J$, $\mathsf{GetP\text{-}c}^{J}_{\downarrow}(p_0)$ returns p_1 later than p_2. The algorithm for CA-$\tau_{\mathcal{R}}$ can be improved as follows.

($\mathit{Enum}^{\mathsf{partial}}\mathit{CA}$-$\tau_{\mathcal{R}}$). $\mathsf{Enum}^{\mathsf{partial}}\mathsf{CA}$-$\tau_{\mathcal{R}}$ calls each $\mathsf{GetP\text{-}c}^{J}_{\downarrow}(p_0)$ in a round-robin fashion, and stops as soon as it has encountered (and processed) a parameter setting that has been returned by all $\mathsf{GetP\text{-}c}^{J}_{\downarrow}(p_0)$'s. This strategy is the same as the one employed by Borzsonyi et al.'s B-tree-based skyline algorithm [Börzsönyi et al. 2001], and has an obvious connection to Fagin's *Threshold Algorithm* [Fagin et al. 2003], which efficiently computes k parameters with the highest aggregated scores given by a monotone function with respect to each criterion.

As in BaseCA-$\tau_{\mathcal{R}}$, $\mathsf{Enum}^{\mathsf{partial}}\mathsf{CA}$-$\tau_{\mathcal{R}}$ maintains the maximal subset A of all qualified parameter settings seen so far. For each qualified parameter setting p (i.e., with $\mathsf{SR}(q(p); r_0) > \tau_{\mathcal{R}}$), $\mathsf{Enum}^{\mathsf{partial}}\mathsf{CA}$-$\tau_{\mathcal{R}}$ adds it to A if $p \not\preceq^{p_0} p'$ for all $p' \in A$. To facilitate this check, we store A in a dynamic spatial index for orthogonal range counting query, for example, range tree. Note that $\mathsf{Enum}^{\mathsf{partial}}\mathsf{CA}$-$\tau_{\mathcal{R}}$ never deletes from A because the order of processing implies that $p' \not\preceq^{p_0} p$ for all $p' \in A$.

Let t denote the final size of the answer set, and let c denote the number of criteria. With $O(\log^c t)$ time for querying and updating the dynamic orthogonal range counting data structure [Chazelle 1988], in the worst case, $\mathsf{Enum}^{\mathsf{partial}}\mathsf{CA}$-$\tau_{\mathcal{R}}$ has time complexity $O(|\mathcal{P}|(\log^c t + \mu_p + \mu_q))$ and space complexity $O(t)$.[9] In practice,

[9]In implementation, instead of using such a dynamic orthogonal range counting data structure, we simply compare a qualified parameter setting against each of the $O(t)$ parameters of A. The worse case time

however, it can perform significantly better than BaseCA-$\tau_{\mathcal{R}}$, because its early stopping condition often results in examining much fewer than $|\mathcal{P}|$ parameter settings. Also, the term t here is the size of the final answer set, whereas the term t in BaseCA-$\tau_{\mathcal{R}}$'s complexity is the maximum size of A during execution, which can be much larger.

For CA-$\tau_{\mathcal{P}}$ and CA-po, recall that a parameter sensibility function SP is required. Suppose that (1) parameter settings can be scored according to each criterion $\jmath$, (2) GetP-$\mathsf{c}^{\jmath}_{\downarrow}(p_0)$ returns them in decreasing score according to $\jmath$, and (3) SP can be defined by a monotone function combining scores for individual criteria. In this case, using Fagin's Threshold Algorithm [Fagin et al. 2003], it is straightforward to implement $\mathsf{GetP}_{\downarrow}(p_0)$ by calling the GetP-$\mathsf{c}^{\jmath}_{\downarrow}(p_0)$'s. With this $\mathsf{GetP}_{\downarrow}(p_0)$, we can then use EnumCA-$\tau_{\mathcal{P}}$ and EnumCA-po (Section 3.2.1) to solve CA-$\tau_{\mathcal{P}}$ and CA-po, respectively.

While the setup of monotone SP in the previous paragraph makes the problem strikingly similar to that in Section 3.2.2, there is a fine but important distinction. Here, each GetP-$\mathsf{c}^{\jmath}_{\downarrow}$ returns (full) parameter settings in $\mathcal{P}$. In contrast, each GetP-$\mathsf{d}^{\iota}_{\downarrow}$ returns only values for a single axis ι of $\mathcal{P}$, and the values must be combined to generate full parameter settings. Therefore, while we can simply use the Threshold Algorithm to implement $\mathsf{GetP}_{\downarrow}$ from GetP-$\mathsf{c}^{\jmath}_{\downarrow}$'s, the same approach does not work for GetP-$\mathsf{d}^{\iota}_{\downarrow}$'s—the algorithm in Section 3.2.2 is needed.

An example application of the $\overset{\text{partial}}{\text{EnumCA}}$-$\tau_{\mathcal{R}}$ algorithm is the TSS-CA$_u$ problem described in Section 5.1. When applying the TSS claim template on the U.S. Congressional voting data, the generator functions GetP-$\mathsf{c}^{\jmath}_{\downarrow}$ correspond the orderings of voter entity u by two criteria, namely, *recognizability* and *ideology*. See Section 5.1 for details.

3.3. The Locus Approach

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 locus 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*[10] in $\mathcal{P}$, whose union covers $\{p \in \mathcal{P} \mid \mathsf{SP}(p; p_0) > \tau_{\mathcal{P}}\}$, the subset of $\mathcal{P}$ with sensibility above $\tau_{\mathcal{P}}$. Given a zone ψ and a reference result r_0, OptP has two variants: $\mathsf{OptP}_{-\infty}(r_0, \psi)$, for CA, returns $\arg\min_{p\in\psi} \mathsf{SR}(q(p); r_0)$, that is, the parameter setting(s) in ψ with minimum result strength relative to r_0; $\mathsf{OptP}_0(r_0, \psi)$, for RE, returns $\arg\min_{p\in\psi} |\mathsf{SR}(q(p); r_0)|$, that is, the parameter setting(s) in ψ whose result is closest to r_0.

---

complexity of $\overset{\text{partial}}{\text{EnumCA}}$-$\tau_{\mathcal{R}}$ becomes $O(|\mathcal{P}|(t + \mu_p + \mu_q))$. With independent axes/criteria, the expected size of A is $O(\log^c |\mathcal{P}|)$ [Buchta 1989].

[10]We leave the definition for a "zone" of $\mathcal{P}$ up to DivP and OptP. 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 for the zone.

Using DivP and OptP$_{-\infty}$, we have the following algorithms for CA. The algorithms for RE are mostly identical, except that they use OptP$_0$ instead of OptP$_{-\infty}$. In the following, let μ_d (and μ_o) denote the running time of DivP (and OptP per zone, respectively).

(*LocCA-$\tau_{\mathcal{P}}$*). Given a parameter sensibility threshold $\tau_{\mathcal{P}}$, LocCA-$\tau_{\mathcal{P}}$ simply calls 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 ψ, LocCA-$\tau_{\mathcal{P}}$ calls OptP(r_0, ψ) to find the best counterarguments. Finally, it returns the overall best counterarguments across all zones.

Let m denote the number of zones returned by DivP. The time complexity of LocCA-$\tau_{\mathcal{P}}$ is $O(\mu_d + m\mu_o)$. The improvement over EnumCA-$\tau_{\mathcal{P}}$ comes from the fact that m is oftentimes far less than the number of parameter settings with sensibility above $\tau_{\mathcal{P}}$; on the other hand, μ_o is likely bigger than μ_q. Thus, the overall savings hinge on the efficiency of OptP.

(*LocCA-$\tau_{\mathcal{R}}$*). This algorithm builds on top of LocCA-$\tau_{\mathcal{P}}$, and is applicable only when a parameter sensibility function SP is available. Given $\tau_{\mathcal{R}}$, we take a guess of the value of $\tau_{\mathcal{P}}$ and call LocCA-$\tau_{\mathcal{P}}$. Suppose it returns an answer with sensibility x and result strength y. We know we have found the desired answer if (1) $y < \tau_{\mathcal{R}}$, and (2) calling LocCA-$\tau_{\mathcal{P}}$ with $\tau_{\mathcal{P}} = x$ would return an answer with result strength no less than $\tau_{\mathcal{R}}$.

We look for the right $\tau_{\mathcal{P}}$ using an exponential search. Starting with a high initial guess for $\tau_{\mathcal{P}}$, we iteratively lower it—effectively doubling the search range for $\tau_{\mathcal{P}}$—until $y > \tau_{\mathcal{R}}$. Then, we perform a binary search in the inferred range of $\tau_{\mathcal{R}}$.

The number of steps in the exponential search is $O(\log |\mathcal{P}|)$, though it can be much smaller in practice when answers have high sensibility. The time complexity of LocCA-$\tau_{\mathcal{R}}$ is thus a factor of $O(\log |\mathcal{P}|)$ higher than LocCA-$\tau_{\mathcal{P}}$.

(*LocCA-po*). This algorithm also builds on top of LocCA-$\tau_{\mathcal{P}}$. We take an initial guess of $\tau_{\mathcal{P}}$, and compute the Pareto-optimal parameter settings (up to k of them) with sensibility higher than $\tau_{\mathcal{P}}$ in decreasing sensibility order. These parameter settings are obtained by calling LocCA-$\tau_{\mathcal{P}}$ repeatedly—first with the initial $\tau_{\mathcal{P}}$, and then subsequently with $\tau_{\mathcal{P}}$ set to the sensibility of the parameter setting returned by the last LocCA-$\tau_{\mathcal{P}}$ call.

We look for the value of $\tau_{\mathcal{P}}$ that gives us the desired number of Pareto-optimal parameter settings using an exponential search. During this search, we take care to avoid repeating LocCA-$\tau_{\mathcal{P}}$ calls with sensibility thresholds that are (effectively) the same.

The algorithm presents the details of the search. Assume that sensibility of all parameter settings fall within the range $(\check{s}_p, \hat{s}_p)$. A^{top} and A^{bot} are lists of $\langle \text{key}, \text{val} \rangle$ pairs, sorted in decreasing key order. Assuming each val is a set, total() returns the total size of all sets in the list, and totalXLast() returns the total size of all but the last set. A^{top} contains all answers with sensibility higher than τ^{top}; A^{bot} contains all answers with sensibility between a guessed threshold $\tau \in (\check{s}_p, \tau^{\text{top}})$ and τ^{top}, right inclusive. The algorithm progressively decreases τ^{top} and increases the number of answers until at least k answers are found. The number of answers returned is capped around k.

The number of steps in the exponential search is $O(\log |\mathcal{P}|)$, though it can be much smaller in practice as explained in the case of LocCA-$\tau_{\mathcal{R}}$ above. The time complexity of LocCA-po is $O(k \log |\mathcal{P}|(\mu_d + m\mu_o))$, where m denotes the maximum number of zones returned by a single DivP call.

We will see instantiations of DivP and OptP in Sections 4.2 and Section 5.2 that would enable the preceding algorithms for claims generalizing our running examples.

ALGORITHM 3: LocCA-po($\langle q, p_0, r_0\rangle, k$).

```
1  A^top ← ∅; τ^top ← ŝ_p; A^bot ← ∅;
2  τ ← initial guess in (š_p, ŝ_p);
3  while τ − š_p > ε do
4  |  A^bot ← ∅; γ ← τ;
5  |  while true do                                   // grow A^bot upwards
6  |  |  (γ′, Δ) ← LocCA-τ_P(⟨q, p_0, r_0⟩, γ);
7  |  |  if Δ = ∅ or γ′ > τ^top then                 // A^bot merges into A^top; start new round
8  |  |  |  A^top ← A^top ∪ A^bot; τ^top ← τ; break;
9  |  |  A^bot.prepend(⟨γ′, Δ⟩);
10 |  |  while A^bot ≠ ∅ and A^top.total() + A^bot.totalXLast() ≥ k do
11 |  |  |  A^bot.removeLast();
12 |  |  γ ← γ′;
13 |  if A^top.total() ≥ k then break;
14 |  τ ← new guess in (š_p, τ);
15 return A^top;
```

4. WAC CLAIMS

Having described our modeling and algorithmic frameworks, we now show how to check a class of claims generalizing Giuliani's in Example 1.1. The generalized claim template can also be applied to other time series data. As an example, we show its application on the U.S. monthly unemployment data in Section 6 (Section 6.1.3 details our choice of model parameters for that application).

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 $\sum_{i\in(t-w,t]} x_i$.[11] The *WAC query template* is the function

$$q(w, t, d) = \frac{\sum_{i\in(t-w,t]} x_i}{\sum_{i\in(t-d-w,t-d]} x_i}, \tag{4}$$

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 $\mathbb{N}^3$ enclosed by 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, for example, "*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" $\mathsf{Nat}(p)$ (independent of p_0) and "relevance" $\mathsf{Rel}(p; p_0)$ (dependent on p_0):

$$\mathsf{SP}(p; p_0) \propto \mathsf{Nat}(p) \cdot \mathsf{Rel}(p; p_0). \tag{5}$$

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 $\mathcal{P}$.

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

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 integers 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 values along an axis of time durations often has a periodic structure, we define naturalness for values for such an axis using a set of (usually a few, and often not disjoint) *levels* whose union is $\mathbb{N}$. Each level ℓ is specified by a pair (χ_ℓ, π_ℓ). Here, χ_ℓ is the naturalness score associated with level ℓ; $\pi_\ell \geq 1$ is an integral period that defines the values in level ℓ as $\mathbb{N}^{(\ell)} = \{v \in \mathbb{N} \mid v \bmod \pi_\ell = 0\}$. The naturalness score of a duration v is given by $\max\{\chi_\ell \mid v \in \mathbb{N}^{(\ell)}\}$; that is, 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

$$\mathsf{Nat}(p) = \mathsf{Nat}_{\mathsf{win}}(w) \cdot \mathsf{Nat}_{\mathsf{lead}}(d), \tag{6}$$

where $\mathsf{Nat}_{\mathsf{win}}$ and $\mathsf{Nat}_{\mathsf{lead}}$ are naturalness scoring functions for the values for the axes of w and d as discussed previously. Specifically, for Giuliani's claim, we define naturalness for both w and d 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 $\mathsf{Rel}(p; p_0)$ to be a 3D normal distribution centered at p_0, that is,

$$\begin{aligned}
\mathsf{Rel}(p; p_0) &= \mathsf{Rel}_{\mathsf{win}}(w; w_0) \cdot \mathsf{Rel}_{\mathsf{tEnd}}(t; t_0) \cdot \mathsf{Rel}_{\mathsf{lead}}(d; d_0), \text{ where} \\
\mathsf{Rel}_{\mathsf{win}}(w; w_0) &= \frac{1}{\sqrt{2\pi}\sigma_{\mathsf{win}}} \exp\left(-\frac{(w - w_0)^2}{2{\sigma_{\mathsf{win}}}^2}\right), \\
\mathsf{Rel}_{\mathsf{tEnd}}(t; t_0) &= \frac{1}{\sqrt{2\pi}\sigma_{\mathsf{tEnd}}} \exp\left(-\frac{(t - t_0)^2}{2{\sigma_{\mathsf{tEnd}}}^2}\right), \\
\mathsf{Rel}_{\mathsf{lead}}(d; d_0) &= \frac{1}{\sqrt{2\pi}\sigma_{\mathsf{lead}}} \exp\left(-\frac{(d - d_0)^2}{2{\sigma_{\mathsf{lead}}}^2}\right).
\end{aligned} \tag{7}$$

In other words, $\mathsf{Rel}(p; p_0) = (2\pi)^{-3/2}|\Sigma|^{-1/2} \exp(-\frac{1}{2}(p - p_0)^T \Sigma^{-1}(p - p_0))$ is the pdf of a 3D normal distribution $\mathcal{N}(p_0; \Sigma)$, where $\Sigma = \mathrm{diag}(\sigma^2_{\mathsf{win}}, \sigma^2_{\mathsf{tEnd}}, \sigma^2_{\mathsf{lead}})$.

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

Recall that Figure 2(b) illustrates the overall parameter sensibility function—the product of naturalness and relevance—for Giuliani's claim when fixing $w = 6$.

Fact-Checking Tasks. The preceding modeling immediately enables the formulation of all problems in Section 2.2 related to finding counterarguments and reverse-engineering for WAC claims. Among the claim quality measures, fairness and robustness are useful to WAC claims in the sense discussed in Example 1.1; uniqueness could be useful in seeing whether the adoption trend existed under other mayors' terms (had more historical data been available).

ALGORITHM 4: GetP-d$^{w}_{\downarrow}(w_0)$ for WAC claims.

Priority queue Q stores triples of the form (v, s, ℓ), where v is the parameter value, s is its contribution to SP and serves as priority, and ℓ is the level (with the highest naturalness score) that v is associated with. Methods add$(\cdot,\cdot,\cdot)$ and removeMax() are self-explanatory.

```
1  def contrib_ℓ(w) begin
2      return χ_ℓ((w − w_0)/σ_w)^2;
3  def next_ℓ(w, increase) begin
4      if increase then
5          if w = w_0 then w' ← w_0 − (w_0 mod π_ℓ) + π_ℓ else w' ← w + π_ℓ;
6      else
7          if w = w_0 then w' ← w_0 − (w_0 mod π_ℓ) else w' ← w − π_ℓ;
8      if w' ∈ [1, n] then return w' else return ⊥;
9  Q ← ∅;
10 foreach level ℓ do
11     foreach increase ∈ {true, false} do
12         w ← next_ℓ(w_0, increase); Q.add(w, contrib_ℓ(w), ℓ);
13 while Q ≠ ∅ do
14     (w, s, ℓ) ← Q.removeMax();
15     w' ← next_ℓ(w, (w > w_0));
16     if w' ≠ ⊥ then Q.add(w', contrib_ℓ(w'), ℓ);
17     yield w;
```

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 $\bar{x}_i = \sum_{j=1}^{i} x_j$ for $i \in [1, n]$. With one pass over the input data, we compute and materialize the prefix sums $\bar{x}_1, \ldots, \bar{x}_n$ in $O(n)$ time. For a given point (w, t, d), the WAC query becomes $q(w, t, d) = \frac{\bar{x}_t - \bar{x}_{t-w}}{\bar{x}_{t-d} - \bar{x}_{t-d-w}}$, which can be computed in $O(1)$ time.

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. Algorithm 2 in Section 3.2.2 can then combine these functions automatically to provide ordered enumeration of full parameter settings.

We show how to implement GetP-d$^{w}_{\downarrow}(w_0)$ that enumerates possible w values with decreasing sensibility with respect to w_0 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. For details, see Algorithm 4.

Ordered enumeration of lead d is analogous. Ordered enumeration 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.

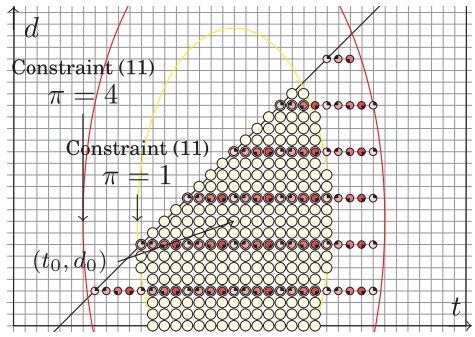


Fig. 4. Illustration of zones $\psi_{w,\ell,h}$ (for a fixed w) dividing the parameter space of WAC claims. The yellow dots (○) belong to the zone for the level with period 1 and the lowest naturalness; the four types of red dots (-◔◑●●), distinguished by their color saturation and orientation, belong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

To understand the complexity of ordered enumeration, we note that all parameter settings above a given sensibility $\tau_{\mathcal{P}}$ fall within the ellipsoid centered at p_0 given by

$$(p - p_0)^T \Sigma^{-1} (p - p_0) = -(2 \ln \tau_{\mathcal{P}} + \ln |\Sigma| + 3 \ln(2\pi)). \tag{8}$$

See Figure 4 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 semiprincipal 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" $\tau_{\mathcal{P}}$, that is, 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. The Locus Approach. We now show how to enable efficient locus approaches, LocCA-$\tau_{\mathcal{P}}$ and LocRE-$\tau_{\mathcal{P}}$, 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 3D 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. Recall that naturalness levels for d-values are specified by $\{(\chi_\ell^d, \pi_\ell^d)\}$, where π_ℓ^d specifies the period of d-values on naturalness level ℓ. Consider the uniform (t, d) grid in the plane with a fixed w-value shown in Figure 4. For a subset of grid points bounded by a convex region, dictated by the problem definition and the minimum sensibility threshold $\tau_{\mathcal{P}}$, we further divide it into π_ℓ^d subsets, with spacing of π_ℓ^d, and offsets $0, 1, \ldots, \pi_\ell^d - 1$ along the t-axis. This further division is for the convenience of OptP as we will show next. Formally, for each $h \in [0, \pi_\ell^d)$, we define such a subset as a zone $\psi_{w,\ell,h}$ as follows:

$$t \leq n \ \wedge \ t - d \geq w; \tag{9}$$

$$d \bmod \pi_\ell^d = 0; \tag{10}$$

ALGORITHM 5: $\mathsf{OptP}_{\text{-}\infty}(r_0, \psi_{w,\ell,h})$ for WAC claims.

Lines 1, 5, and 6 optimize under Constraints (9), (10), (11)*, and* (12)*; details are omitted.*
// Determine the range of time points:
1 $i^{\min} \leftarrow \min\{t - d \mid (w,t,d) \in \psi_{w,\ell,h}\}$; $i^{\max} \leftarrow \max\{t \mid (w,t,d) \in \psi_{w,\ell,h}\}$;
2 **for** $(i \leftarrow i^{\min}; i \le i^{\max}; i \leftarrow i + \pi^d_\ell)$ **do** $y_i \leftarrow \bar{x}_i - \bar{x}_{i-w}$;
3 $\mathcal{Y} \leftarrow$ a data structure for $\{y_i\}$ supporting range-maximum queries;
// Search the zone, one t value at a time:
4 $(t^\star, d^\star) \leftarrow (\perp, \perp)$; $s^\star \leftarrow \infty$;
5 **foreach** $i_2 \in \{t \mid \exists d : (w,t,d) \in \psi_{w,\ell,h}\}$ **do**
6 $\quad i_1^{\min} \leftarrow \min\{i_2 - d \mid (w, i_2, d) \in \psi_{w,\ell,h}\}$; $i_1^{\max} \leftarrow \max\{i_2 - d \mid (w, i_2, d) \in \psi_{w,\ell,h}\}$;
7 $\quad (i_1, y_{i_1}) \leftarrow \mathcal{Y}.\mathsf{range\text{-}max}([i_1^{\min}, i_1^{\max}])$;
8 $\quad s \leftarrow \frac{y_{i_2}}{y_{i_1}} / r_0 - 1$;
9 $\quad$ **if** $s < s^\star$ **then**
10 $\quad\quad (t^\star, d^\star) \leftarrow (i_2, i_2 - i_1)$; $s^\star \leftarrow s$;
11 **return** $(w, t^\star, d^\star)$;

$$\left(\frac{t - t_0}{\sigma_t}\right)^2 + \left(\frac{d - d_0}{\sigma_d}\right)^2 < -\ln \tau, \text{ where } \tau = \frac{\tau_{\mathcal{P}}}{\mathsf{Nat}_{\mathsf{win}}(w) \cdot \mathsf{Rel}_{\mathsf{win}}(w; w_0) \cdot \chi^d_\ell}; \tag{11}$$

$$t \bmod \pi^d_\ell = h. \tag{12}$$

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 in the following). 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 1D problem. However, not all combinations of (t,d) are valid according to the semantics of WAC claim. Constraint (12) 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 in the following.

For example, in Giuliani's example where the three naturalness levels $\{(\chi^d_\ell, \pi^d_\ell)\}_{\ell=1}^3 = \{(1,1), (e,4), (e^2, 8)\}$ are defined for d, DivP returns, for all parameters on the lowest naturalness level $\ell = 1$ (within the plausible region defined by Equations (9) and (11), a single zone $\psi_{w,1,0}$. For $\ell = 2$, where $\pi^d_\ell = 4$, four zones are returned, denoted by $\psi_{w,2,h}$, $h = 0, 1, 2, 3$. Each zone contains parameters on naturalness level $\ell = 2$, with the additional constraint that $t \bmod 4 = h$; similar for level $\ell = 3$ with $\pi^d_3 = 8$ zones returned by DivP.

Optimize: CA. Next, we show how to optimize each zone returned by DivP. For the problem of finding counterarguments, we define $\mathsf{OptP}_{\text{-}\infty}$ for WAC (Algorithm 5) as follows. Note that for each zone $\psi_{w,\ell,h}$ there exists an equally spaced subsequence of time points $I = \{i \mid i \bmod \pi^d_\ell = h \wedge i^{\min} \le i \le i^{\max}\}$ (where $i^{\min}$ and $i^{\max}$ are derived from the constraints on t and d imposed by the zone), such that for every (t,d) setting in $\psi_{w,\ell,h}$, both $t \bmod \pi^d_\ell = h$ and $t - d \bmod \pi^d_\ell = h$. We compute the window aggregate result (with window length w) for each of these time points in I, 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 values, namely, $(i_1, i_2) = (t - d, t)$; we call such pairs *valid*. Note that $\mathsf{SR}((w,t,d); r_0) = \frac{1}{r_0} \cdot \frac{y_{i_2}}{y_{i_1}} - 1$.

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

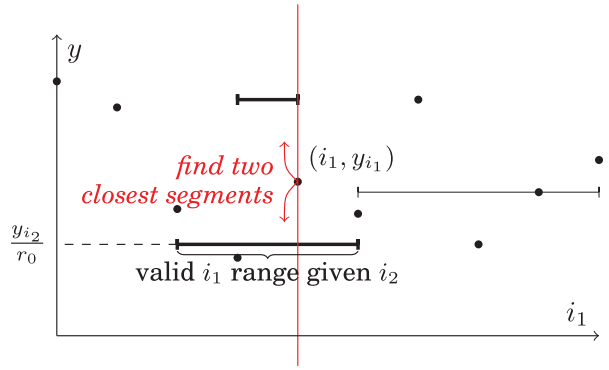


Fig. 5. Illustration of the sweep line procedure used by OptP_0 for reverse-engineering WAC claims. Points corresponding to Y are drawn as black dots. Segments currently intersected by the (red) sweepline are drawn using thick lines.

reduces to a range-maximum query over a (static) sequence, a well-studied problem. The sequence Y can be preprocessed in linear ($O(|Y|)$) time into a linear-space data structure, such that any range-maximum query can be answered in $O(1)$ time [Harel and Tarjan 1984].[12] We use the same notion of "interesting solution radius" $\tilde{r}$ introduced in Section 4.2.2 to analyze the complexity of the locus approach for WAC with the DivP and $\mathsf{OptP}_{-\infty}$ 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.

Note that complexity analysis discussed previously is for LocCA-$\tau_{\mathcal{P}}$. For the other two versions, as mentioned in Section 3.3, LocCA-$\tau_{\mathcal{R}}$ invokes LocCA-$\tau_{\mathcal{P}}$, within an additional multiplicative factor of $O(\log |\mathcal{P}|)$ in its time complexity, that is, $O(\tilde{r}^2 \log n)$. For LocCA-po, with another multiplicative factor of $O(k)$, the time complexity is $O(k\tilde{r}^2 \log n)$.

Optimize: 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 $\mathsf{OptP}_{-\infty}$, 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 $\frac{1}{r_0} \cdot \frac{y_{i_2}}{y_{i_1}} - 1$ over all valid (i_1, i_2) pairs as in $\mathsf{OptP}_{-\infty}$, we want to minimize the absolute result strength difference $|\frac{1}{r_0} \cdot \frac{y_{i_2}}{y_{i_1}} - 1|$, or equivalently, $|\frac{1}{y_{i_1}}| \cdot |y_{i_1} - \frac{y_{i_2}}{r_0}|$.

Given a valid i_2 value, we can determine the range of valid i_1 values associated with i_2, as in $\mathsf{OptP}_{-\infty}$. Recall that $\mathsf{OptP}_{-\infty}$ 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 2D as shown in Figure 5. 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 $\frac{y_{i_2}}{r_0}$. It is easy to see that the i_1 value that minimizes $|\frac{1}{r_0} \cdot \frac{y_{i_2}}{y_{i_1}} - 1|$ 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

[12]We actually use a simple implementation based on Tarjan's offline LCA algorithm [Tarjan 1979]. 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.

by the horizontal coordinates of their endpoints. We sweep a vertical line from left to right. During the sweep, we incrementally maintain the set of segments intersected by the sweep line in a 1D search tree (e.g., B-tree) with the height of a segment as its key. For each point (i, y_i) encountered by the sweep line, 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 $|\frac{1}{r_0} \cdot \frac{y_{i_2}}{y_{i_1}} - 1|$) 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 of which takes $O(\log |Y|)$ 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.

Similar to CA, the time complexities of LocRE-$\tau_{\mathcal{R}}$ and LocRE-po are factors of $O(\log |\mathcal{P}|)$ and $O(k \log |\mathcal{P}|)$ higher than that of LocRE-$\tau_{\mathcal{P}}$, respectively, hence $O(\tilde{r}^2 \log \tilde{r} \log n)$ and $O(k\tilde{r}^2 \log \tilde{r} \log n)$.

5. TSS: TIME SERIES SIMILARITY CLAIMS

We now turn to a class of claims generalizing the vote correlation claim in Example 1.2.

5.1. Modeling TSS

Parameterized Query Template. Here the database contains information about m entities identified as $1, 2, \ldots, m$. Each entity u is associated with a time series $X_u = \{x_{u,1}, x_{u,2}, \ldots, x_{u,n}\}$. A function $\mathrm{sim}_T(X_u, X_v)$, where $T \subseteq [1, n]$ and $u, v \in [1, m]$, computes the similarity between two time series $\{x_{u,t} \in X_u \mid t \in T\}$ and $\{x_{v,t} \in X_v \mid t \in T\}$, that is, X_u and X_v restricted to the subset of timesteps in T. The *TSSquery template* is the function

$$q(u, v, a, b) = \mathrm{sim}_{[a,b]}(X_u, X_v), \tag{13}$$

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 1.2, 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. Since different problem formulations call for different setups for parameter sensibility and result strength, we organize our discussion below by problem formulation.

Finding Counterarguments by Perturbing Comparison Period (*TSS-CA$_{ab}$*). Here, we fix u and v to those in the original claim, and consider counterarguments obtained by perturbing a and b. We call this problem *TSS-CA$_{ab}$*. Suppose higher similarity strengthens 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 Equation (5).

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 $\mathbb{N}$. However, we do not require levels to be periodic in this case. In general, each level ℓ is specified by a pair $(\chi_\ell, \lambda_\ell)$, where χ_ℓ is the naturalness score associated with level ℓ and $\lambda_\ell : \mathbb{N} \to \{\texttt{false}, \texttt{true}\}$ is a condition that "selects" the values in level ℓ. Again, the naturalness score of a time point t is given by $\max\{\chi_\ell \mid \lambda_\ell(t) = \texttt{true}\}$; that is, the maximum score that v is associated with. Specifically, for the Marshall-Boehner claim, we define naturalness of a using two levels $(1, t \mapsto \texttt{true})$ and $(e, t \mapsto t \in \mathbb{N}^{\mathsf{begin}})$, where $\mathbb{N}^{\mathsf{begin}}$ is the subset of $[1, n]$ corresponding to the beginning of some session of Congress.

The relevance of $p = (a, b)$ relative to $p_0(a_0, b_0)$ decreases with the distance between them, and is defined analogously to WAC claims in Equation (7), Section 4.1:

$$\mathsf{Rel}_{\mathsf{time}}(p; p_0) = (2\pi)^{-1}|\Sigma|^{-1/2} \exp\left(\frac{1}{2}(p - p_0)^T \Sigma^{-1}(p - p_0)\right), \quad \text{where } \Sigma = \mathrm{diag}({\sigma_{\mathsf{begin}}}^2, {\sigma_{\mathsf{end}}}^2). \tag{14}$$

Specifically, for the Marshall-Boehner claim, $(\sigma_{\mathsf{begin}}, \sigma_{\mathsf{end}}) = (1000, 1)$. We use a small σ_{end} to penalize perturbation in b, because its original setting reflects the time of the claim.

With the preceding definitions, we obtain the three variants of the problem of finding counterarguments in Section 2.2, for perturbations of the comparison time period.

Finding Counterarguments by Perturbing Entities (TSS-CA$_u$). Now, consider counterarguments obtained by perturbing entity parameters, while fixing a and b (to their settings from the original claim, a counterargument obtained from TSS-CA$_{ab}$, or a reverse-engineered claim as discussed later). Replacing both u and v would lead to settings with strenuous connection to the original claim, so we consider perturbing u only (as in the counterarguments made by professional fact checkers at factcheck.org in Example 1.2).[13] We call this problem *TSS-CA$_u$*.

Intuitively, we want to find an entity u that is recognizable to the audience, yet does not have the property that the original claim tries to suggest for u_0. For example, the property suggested by the Marshall-Boehner claim is "being conservative," so to counter it, we choose a well-known Democrat Jim Clyburn. In general, we model the aspects of recognizability and (deviation from) suggested property as naturalness $\mathsf{Nat}_{\mathsf{src}}(u)$ and relevancy $\mathsf{Nat}_{\mathsf{src}}(u)$, respectively. Instead of combining the two scores into a parameter sensibility function, we use a parameter sensibility relation $\preceq^{u_0}$ to define a partial order: $u_1 \preceq^{u_0} u_2$ if and only if $\mathsf{Nat}_{\mathsf{src}}(u_1) \leq \mathsf{Nat}_{\mathsf{src}}(u_2)$ and $\mathsf{Nat}_{\mathsf{src}}(u_1; u_0) \leq \mathsf{Nat}_{\mathsf{src}}(u_2; u_0)$.

Specifically, for the vote correlation claim, we assume that each Representative e is annotated with a recognizability score $e.\texttt{rec} \in \mathbb{R}$ (higher means more recognizable) and an ideology score $e.\texttt{ide} \in \mathbb{R}$ (higher means more conservative); see Section 6 for how we obtain these annotations. We define

$$\mathsf{Nat}_{\mathsf{src}}(u) = u.\texttt{rec}; \tag{15}$$

$$\mathsf{Nat}_{\mathsf{src}}(u; u_0) = u_0.\texttt{ide} - u.\texttt{ide}. \tag{16}$$

Note that $\mathsf{Nat}_{\mathsf{src}}(e; u_0)$ is defined such that liberal Representatives will be scored higher, because for this example, we want to use such Representatives to counter the claim that high vote correlation with Boehner implies being conservative.

[13]It is also plausible to consider perturbing v alone; for example, we could replace Boehner in the vote correlation claim with a leading Democrat to argue that Marshall also votes with liberals. We omit the discussion here as the problem formulation changes only slightly.

As for result strength, note that a Representative used to counter the claim should have reasonable (relative to Marshall) vote correlation with Boehner—the higher the better. Therefore, we define the result strength function as $\mathsf{SR}(r; r_0) = r_0 - r$ (note the reversal from the case of TSS-CA$_{ab}$). In other words, higher vote correlations in this case lead to better counterarguments that weaken the original claim more.

With the preceding definitions, we obtain variant CA-$\tau_{\mathcal{R}}$ of the problem of finding counterarguments for perturbations of the source entity.

Reverse-Engineering Vague Claims. TSS claims always mention u explicitly, so for reverse-engineering, we only consider settings of a, b, and v. For example, in the original claim of Example 1.2, u is Marshall; v is vaguely stated as "Republican leaders"; a and b are omitted. Suppose $\mathsf{SP}_{\mathsf{tgt}}(v)$ captures how "sensible" v is in the context of the original claim. We then define the overall parameter sensibility function by multiplying this v-based score with the time-based scores defined earlier for TSS-CA$_{ab}$:

$$\mathsf{SP}\left((v, a, b); (v_0, a_0, b_0)\right) = \mathsf{SP}_{\mathsf{tgt}}(v) \cdot \mathsf{Nat}_{\mathsf{begin}}(a) \cdot \mathsf{Rel}_{\mathsf{begin}}(a; a_0) \cdot \mathsf{Rel}_{\mathsf{end}}(b; b_0). \tag{17}$$

Given the nature of vague TSS claims, choices of v are often limited; such is the case for "Republican leaders" in Example 1.2. Suppose there are η "sensible" choices of v. We can simply define $\mathsf{SP}_{\mathsf{tgt}}(v) = 1/\eta$ for all sensible choices of v, and 0 for other entities. For the specific definition of "sensible choices" for the vote correlation claim, see Section 6.

We define the result strength as $\mathsf{SR}(r; r_0) = r - r_0$, which simply reflects the difference between results (reversing the direction of subtraction would make no practical difference).

These definitions give us all variants of the reverse-engineering problem discussed in Section 2.2.

Measuring Claim Quality. Claim qualities can be measured by perturbing either the comparison period or the entities. First, consider the perturbations of a and b while fixing the original u and v. We use the result strength and parameter sensibility functions defined earlier for TSS-CA$_{ab}$ (SP needs to be normalized to a pmf). Among the claim quality measures defined in Section 2.2, both fairness and robustness are useful in this case; uniqueness is not.

Second, considering other settings of u and v gives us a sense of how special the degree of similarity in the original claim is. (Fixing either u or v, instead of both, also leads to meaningful formulations.) Here, we define the parameter space $\mathcal{P}$ to be all possible (u, v) pairs (unique up to order) where $u \neq v$. In this case, uniqueness makes obvious sense, when we define $\mathsf{SR}(r; r_0) = r - r_0$ (assuming higher similarity strengthens the claim).

In the second case mentioned earlier, we can further define a parameter sensibility function that assigns equal probability to each $(u, v) \in \mathcal{P}$. With this pmf, fairness becomes a useful measure as well. It computes the average relative strength of perturbed claims; a positive or small negative value means the claimed similarity is actually not high in comparison.

Other Applications. TSS claims can be applied to other multitime series datasets, such as stock prices over time, and sports players' game-by-game performance stats. We do not detail the other applications in this article.

5.2. Algorithmic Building Blocks for TSS

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* [Gray et al. 1996] over the set $\{\mathrm{sim}_{[t,t]}(X_u, X_v) \mid t \in [a, b]\}$ of

pointwise 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.

For TSS-CA$_u$, we are given the time period $[a, b]$ and the target entity v, but need to permute the source entity u. In this case, precomputing the queries for different source entities brings no benefit compared with computing them on demand. However, in the preprocessing step, we cache the portion of X_v during $[a, b]$ in memory, so any subsequent queries comparing X_u with X_v only needs to retrieve the corresponding portion of X_u.

For TSS-RE, we need to find the best (v, a, b) settings while u is fixed. In this case, for each sensible choice of v, we perform the same preprocessing as in TSS-CA$_{ab}$, so that we can compute each query in $O(1)$ time.

5.2.2. Ordered Enumeration of Parameters. For TSS-CA$_{ab}$, enabling ordered enumeration of (a, b) settings is straightforward—we follow the same approach as for WAC claims in Section 4.2.2, that is, by providing a function to enumerate values in each axis of the parameter space in order. Since b contributes only to naturalness, we simply enumerate b values in increasing distance from b_0. a contributes to both naturalness and relevance, and can be handled in a way similar to w for WAC claims in Section 4.2.2—enumerating values across all levels of naturalness in parallel and merging them using a priority queue. The only modification required to Algorithm 4 is a slight generalization of the procedure next_ℓ: here, instead of returning the next value divisible by period π_ℓ of level ℓ, we return the next value satisfying the condition λ_ℓ for level ℓ. Finally, we define $\mathsf{IsPValid}$ to ensure that $a \le b$. Following the notion of interesting solution radius $\tilde{r}$ introduced in Section 4.2.2, the time and space complexities are $O(\tilde{r}^2 \log \tilde{r})$ and $O(\tilde{r})$, respectively.

For TSS-CA$_u$, the source entity parameter u needs to be permuted, and its values are partially ordered by the two criteria of naturalness and relevance. In the case of vote correlation claims, these criteria are recognizability (Equation (15)) and ideology (Equation (16)), respectively. It is straightforward to provide a function for enumerating entities by each criterion. Then, as discussed in Section 3.2.3, algorithm $\mathsf{Enum}^{\mathrm{partial}}\mathsf{CA}\text{-}\tau_{\mathcal{R}}$ immediately becomes applicable.

Finally, for TSS-RE, we extend the method for enumerating (a, b) described previously for TSS-CA$_{ab}$ with the additional axis for target entity v. In addition to the already defined functions for enumerating a and b, we simply add a function that enumerates v in decreasing order of $\mathsf{SP}_{\mathsf{tgt}}(v)$ (recall Equation (17)). For the Marshall-Boehner claim, this function simply enumerates all Representatives who can be regarded as "Republican leaders."

5.2.3. The Locus Approach. We now describe how to enable the locus approach for TSS-CA$_{ab}$ and TSS-RE (this approach is not applicable to TSS-CA$_u$). At 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.

TSS-CA$_{ab}$. Given a $p_0 = (a_0, b_0)$ and a sensibility threshold $\tau_{\mathcal{P}}$, we want to compute $p = (a, b)$ that minimizes $\mathsf{SR}(q(p); q(p_0))$, where p satisfies $\mathsf{SP}(p; p_0) > \tau_{\mathcal{P}}$.

Observe that the subset of (a, b) parameter settings above $\tau_{\mathcal{P}}$ is a set of 2D grid points. This subset is analogous to a slice of the 3D grid points (with w fixed) in Section 4.2.3, but further division into zones is simpler in this case. For each naturalness level ℓ along

the a-axis specified by $(\chi_\ell, \lambda_\ell)$, we let DivP return zone ψ_ℓ defined by the constraints below[14]:

$$1 \le a \le b \le n; \tag{18}$$

$$\lambda_\ell(a) = \texttt{true}; \tag{19}$$

$$\left(\frac{a-a_0}{\sigma_a}\right)^2 + \left(\frac{b-b_0}{\sigma_b}\right)^2 < -\ln\frac{\tau_{\mathcal{P}}}{\chi_\ell}. \tag{20}$$

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 (18) and (20); the high-naturalness zone is a subset of the grid points with $a \in \mathbb{N}^{\mathsf{begin}}$.

OptP$_{-\infty}$ for TSS-CA$_{ab}$ can be formulated as follows. For a given $a \in [1, n]$, let $J_a \subseteq [1, n]$ denote the interval of b values that satisfy Constraints (18) and (20). Given a zone ψ_ℓ, for each value of a that is valid in ψ_ℓ, we compute

$$b_a^\star = \arg\min_{b \in J_a} \frac{\bar{s}_b - \bar{s}_{a-1}}{\bar{c}_b - \bar{c}_{a-1}}. \tag{21}$$

Intuitively, the objective is to minimize the slope of the line from o_{a-1} to $o_b \in J_a$. We describe a data structure for computing $b_a^\star$ for a given value of a.

Let $\mathcal{O} = \{o_i = (\bar{c}_i, \bar{s}_i) \mid 0 \le i \le n\}$ be a set of points in $\mathbb{R}^2$ representing the pairs formed by the corresponding elements of the two series $\{\bar{c}_i\}$ and $\{\bar{s}_i\}$ obtained by the preprocessing step in Section 5.2.1. For simplicity, we assume that $n = 2^k$ for some integer $k \ge 0$. Since $\bar{c}_{i+1} \ge \bar{c}_i$ and $\bar{s}_{i+1} \ge \bar{s}_i$, the points in $\mathcal{O}$ form an xy-monotone chain. For a subset $X \subseteq \mathcal{O}$ and a value $a \in [1, n]$, let

$$\mathcal{Q}(X, a) = \arg\min_{(\bar{c}_b, \bar{s}_b) \in X} \frac{\bar{s}_b - \bar{s}_{a-1}}{\bar{c}_b - \bar{c}_{a-1}}. \tag{22}$$

As an auxiliary function, $\mathcal{Q}(X, a)$ denotes the point in X that minimizes the slope of the line formed together with o_{a-1}. Let $\mathcal{O}_a = \{o_b \mid b \in J_a\}$. Then $\mathcal{Q}(\mathcal{O}_a, a) = o_{b_a^\star}$. It thus suffices to describe how to compute $\mathcal{Q}(\mathcal{O}_a, a)$. The line connecting o_{a-1} and $\mathcal{Q}(\mathcal{O}_a, a)$ must be tangent to the lower boundary of the convex hull of X. This observation leads us to the following data structure supporting fast computation of the lower boundary of convex hulls over different ranges of points in $\mathcal{O}$.

We construct a 1D quad tree $\mathbb{T}$ on the interval $(0, n]$, a complete binary tree with n leaves whose i-th leaf is associated with the interval $(i-1, i]$. Each node θ of $\mathbb{T}$ is associated with a *canonical interval* γ_θ such that for an internal node θ with children ζ and ξ, $\gamma_\theta = \gamma_\zeta \cup \gamma_\xi$; the root interval $\gamma_{\text{root}} = (0, n]$. Let $O^\theta = \{o_i \mid i \in \gamma_\theta\}$. We construct LH_θ, the lower boundary of the convex hull of O^θ, using LH_ζ and LH_ξ, in $O(|O^\theta|)$ time [De Berg et al. 2000]. Figure 6 illustrates an example of this data structure (ignore the blue portions for now). We store the sequence of vertices of LH_θ in an array, augmented with the fractional-cascading data structure [De Berg et al. 2000]. The fractional-cascading structure enables us to perform binary search at nodes along a path in $O(1)$ time per node, after spending $O(\log n)$ time at the root (see De Berg et al. [2000] for details).

Given $a \in [1, n]$, we compute $\mathcal{C}_a$, the (unique) canonical interval decomposition of J_a on $\mathbb{T}$. It is well known that $\mathcal{C}_a$ consists of $O(\log n)$ nodes with disjoint canonical intervals, whose union is J_a. It is easy to see that $\mathcal{O}_a = \bigcup_{\theta \in \mathcal{C}_a} O^\theta$, so $\mathcal{Q}(\mathcal{O}_a, a) = \mathcal{Q}(\{\mathcal{Q}(O^\theta, a) \mid \theta \in \mathcal{C}_a\}, a)$, that is, the optimal solution for a on J_a comes from the optimal solutions for a on one of the canonical intervals in $\mathcal{C}_a$. Fix a node $\theta \in \mathcal{C}_a$, since the line passing through

[14]Narrow windows do not make sense. The model and algorithm can be adapted to penalize for this.

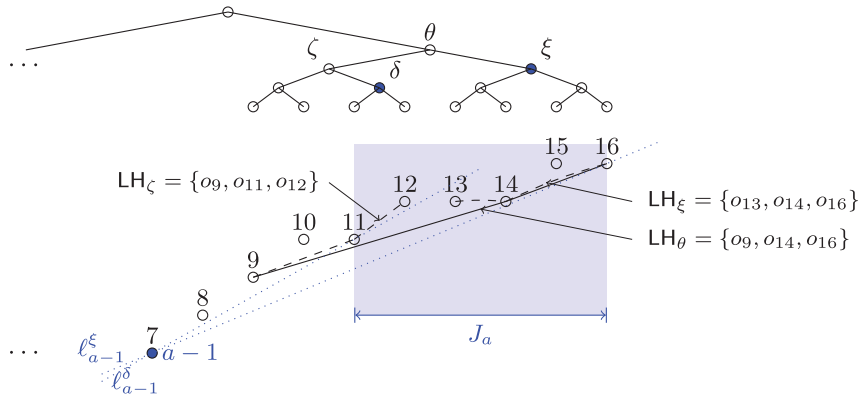


Fig. 6. Illustration of the locus approach for TSS-CA$_{ab}$. The two dashed chains show LH_ζ and LH_ξ; the solid chain shows LH_θ. Suppose $a = 8$ and $J_a = [11, 16]$. Then $\mathcal{C}_a = \{\delta, \xi\}$ (note that $\mathsf{LH}_\delta = \{o_{11}, o_{12}\}$ is not labeled in the figure). Since ℓ^ξ_{a-1} has smaller slope than ℓ^δ_{a-1}, we have $b^\star_a = 16$ and $\mathcal{Q}(\mathcal{O}_8, 8) = o_{16}$.

o_{a-1} and $\mathcal{Q}(O^\theta, a)$, denoted by ℓ^θ_{a-1}, has the minimum slope among all lines passing through o_{a-1} and a point of O^θ, LH_θ lies above ℓ^θ_{a-1} and thus ℓ^θ_{a-1} is tangent to LH_θ. We can thus find $\mathcal{Q}(O^\theta, a)$ by doing a binary search over the vertices of LH_θ. See (the blue portions of) Figure 6 for an illustration. Note that the nodes in $\mathcal{C}_a$ lie along two paths of $\mathbb{T}$. Using the fractional-cascading structure, we can compute $\mathcal{Q}(O^\theta, a)$ for all $\theta \in \mathcal{C}_a$ in a total of $O(\log n)$ time. Hence, $b^\star_a$ can be computed in $O(\log n)$ time.

By querying $\mathbb{T}$ with all values of a that are valid for ψ_ℓ, and repeating this procedure for all zones, OptP_∞ returns $\arg\min_{p \in \psi_\ell} \mathsf{SR}(q(p); q(p_0))$ in $O(n \log n)$ time. For TSS-CA$_{ab}$, the number of zones is the same as the number of naturalness levels, which is a domain-specific constant. So the overall time complexity is also $O(n \log n)$, or $O(\tilde{r} \log \tilde{r})$, following the notion of interesting solution radius.

TSS-RE. In this case, the space of (v, a, b) parameter settings has one more axis v than the previous case of TSS-CA$_{ab}$. Given a $p_0 = (v_0, a_0, b_0)$, a reference response r_0, and a sensibility threshold $\tau_\mathcal{P}$, the goal is to compute $p = (v, a, b)$ that minimizes $|\mathsf{SR}(q(p); r_0)|$, where p satisifies $\mathsf{SP}(p; p_0) > \tau_\mathcal{P}$.

We slice the subset of parameter settings with sensibility above $\tau_\mathcal{P}$ by all possible values of v, and then further divide each slice into zones in a way similar to the case of TSS-CA$_{ab}$. More precisely, for each v with nonzero $\mathsf{SP}_{\mathsf{tgt}}(v)$ and for each naturalness level ℓ along the a axis, we let DivP return zone $\psi_{v,\ell}$, which is defined by Constraints (18) and (19), as well as the following (which replaces Constraint (20)):

$$\left(\frac{a - a_0}{\sigma_a}\right)^2 + \left(\frac{b - b_0}{\sigma_b}\right)^2 < -\ln \frac{\tau_\mathcal{P}}{\mathsf{SP}_{\mathsf{tgt}}(v) \cdot \chi_\ell}. \tag{23}$$

Given zone $\psi_{v,\ell}$ and reference result r_0, we define $\mathsf{OptP}_0(r_0, v, \ell)$ as follows. For a value a, let J_a be the interval of b values that satisfy (18) and (20), and let

$$\hat{b}_a = \arg\min_{b \in J_a} \left| \frac{\bar{s}_b - \bar{s}_{a-1}}{\bar{c}_b - \bar{c}_{a-1}} - r_0 \right|. \tag{24}$$

Then the goal is to find the pair $(a, \hat{b}_a)$, over all valid values of a in $\psi_{v,\ell}$, such that $|\frac{\bar{s}_b - \bar{s}_{a-1}}{\bar{c}_b - \bar{c}_{a-1}} - r_0|$ is minimized.

We adapt the algorithm for TSS-CA$_{ab}$. Let $\mathcal{O}$ and $\mathcal{O}_a$ be the same as for the TSS-CA$_{ab}$ case. For a given $a \in [1, n]$, let ℓ_{a-1} be the line of slope r_0 passing through o_{a-1}, and let

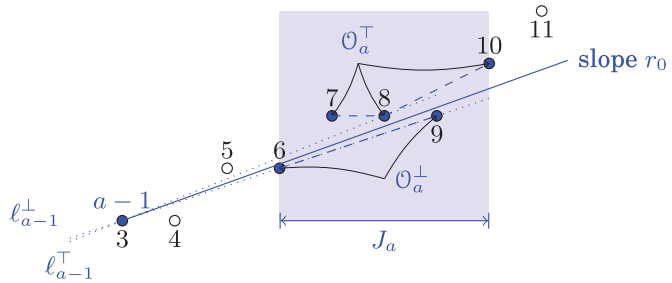


Fig. 7. Illustration of the locus approach for TSS-RE. Suppose $a = 4$ and $J_a = [6, 10]$. The line of slope r_0 passes through o_{a-1}, and divides the points in range J_a into $\mathcal{O}_a^\top$ and $\mathcal{O}_a^\perp$. The lower and upper convex hulls of $\mathcal{O}_a^\top$ and $\mathcal{O}_a^\perp$ are shown as dashed chains. Dotted line $\ell_{a-1}^\top$ ($\ell_{a-1}^\perp$, respectively) passes through o_{a-1} and $b_a^\top = o_8$ ($b_a^\perp = o_6$, respectively). Because $\ell_{a-1}^\perp$ is closer to target slope, $\hat{b}_a = 6$.

$\mathcal{O}_a^\top$ ($\mathcal{O}_a^\perp$, respectively) be the subset of $\mathcal{O}_a$ of the points lying above (below, respectively) ℓ_{a-1}, and let $b_a^\top$ ($b_a^\perp$, respectively) be the index such that the line $\ell_{a-1}^\top$ ($\ell_{a-1}^\perp$, respectively) passing through o_{a-1} and $o_{b_a^\top}$ ($o_{b_a^\perp}$, respectively) has the smallest (largest, respectively) slope among all lines passing through o_{a-1} and a point of $\mathcal{O}_a^\top$ ($\mathcal{O}_a^\perp$, respectively). Then, $\hat{b}_a = b_a^\top$ if ℓ_{a-1} makes a smaller angle with $\ell_{a-1}^\top$ than with $\ell_{a-1}^\perp$, and $\hat{b}_a = b_a^\perp$ otherwise. Figure 7 illustrates this intuition.

We now describe how to adapt the data structure for TSS-CA$_{ab}$ to compute $b_a^\top$ for all valid a values in the case of TSS-RE ($b_a^\perp$ can be computed in an analogous manner). There are two main differences in the data structure: (i) we process the a values in a specific order, and (ii) we build a semidynamic quad tree $\mathbb{T}$ in which the points in $\mathcal{O}$ are inserted one by one. More precisely, let ℓ_0 be the line of slope r_0 passing through the origin, and let u_0 be the unit vector normal to ℓ_0 and lying in the half-plane below ℓ_0. We process the a values in increasing order of the dot product $\langle o_a, u_0 \rangle$. In other words, we sweep a line of slope r_0 from top to bottom, and process the points in $\mathcal{O}$ as the line sweeps across them.

As in TSS-CA$_{ab}$, we use a 1D quad tree $\mathbb{T}$ whose nodes store the lower convex hulls of the points in the corresponding canonical ranges. We augment $\mathbb{T}$ with the semidynamic fractional-cascading structure [Mehlhorn and Näher 1990]. During the sweep, we maintain $\mathbb{T}$ such that it indexes all points above the sweep line. Suppose we now encounter point o_{a-1}. At this moment, $\mathbb{T}$ indexes the subset $\mathcal{O}'$ of the points above ℓ_{a-1}; each node $\theta \in \mathbb{T}$ stores LH_θ, the lower hull of $\mathcal{O}' \cap \gamma_\theta$. We compute $b_a^\top$ by querying $\mathbb{T}$ with J_a as in the algorithm for TSS-CA$_{ab}$. After computing $b_a^\top$, we insert o_{a-1} into $\mathbb{T}$, which updates the lower convex hulls and the fractional-cascading structure. The amortized time spent in computing $b_a^\top$ and inserting o_{a-1} is $O(\log n)$. Hence, the total time spent by the algorithm is $O(n \log n)$.

The total time spent on all zones of a fixed entity v is the same as that of TSS-CA$_{ab}$, that is, $O(n \log n)$. Repeating this process for all m entities takes $O(mn \log n)$ time. Again, using the notion of interesting solution radius, the time complexity becomes $O(m\tilde{r} \log \tilde{r})$.

6. EXPERIMENTS

Our experiments serve two purposes. First, we begin with proof-of-concept experiments that apply our QRS framework to real-life claims and datasets, and illustrate the usefulness of our results. Then, we demonstrate the efficiency and scalability of our algorithms, showing how enumeration and locus 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 New York City, we use the following datasets: ***UNEMP*** records the U.S. monthly unemployment rate for 782 months from January 1948 to February 2013[15]; ***AUTO*** contains daily auto accident statistics in Texas from 2003 to 2011 (with 3,287 data points)[16]; ***VOTE*** contains 22.32 million votes cast by 12,572 members of the U.S. Congress from May 1789 to April 2013.[17]

Besides the evaluation of algorithms in this section, we refer interested readers to Wu et al. [2014], a demonstration-track paper describing a system prototype implementing our framework and algorithms in a hierarchical and extensible way.

6.1. Proof of Concept

We first show the quality of results of solving the reverse-engineering and the counterargument finding problems within the QRS framework for both the WAC and the TSS claims. We apply the WAC claim template on the NYC adoption data (as described in Section 4) and the UNEMP data, and then the TSS claim template on the VOTE data (as described in Section 5).

6.1.1. Giuliani's Adoption Claim. We first use our technique to reverse-engineer Giuliani's vague adoption claim in Example 1.1. Recall the model in Section 4.1. As discussed in Section 2.2, we set $p_0 = (1, 2001, 8)$, which captures the claim context that Giuliani served the 8-year term 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, that is, Giuliani's first and second 4-year terms. This counterargument leads to a 1% decrease in the adoption rate (as opposed to the big increase in the original claim), exposing the actual trend after Giuliani took office.

6.1.2. Marshall's Vote Correlation Claim. As another proof of concept, consider Marshall's vote correlation claim in Example 1.2. Recall the model in Section 5.1. For the recognizability score, we use a rather crude measure—the length (in bytes) of a Representative's Wikipedia page. For the ideology score, we use the well-known *NOMINATE* method.[18]

To reverse-engineer the original claim, we set up its context as follows. We let b_0 be the time when the claim was made (Oct. 2010), and a_0 be the start of the term in which the claim was made (Jan. 2010). We define the search space for the "Republican leader" as the five most recognizable Republican Representatives active as of b_0 (Ron Paul, Paul Ryan, Michele Bachmann, John Boehner, and Aaron Schock). We set u to be Marshall and $r_0 = 0.65$ as stated in the claim. We then invoke our algorithm for TSS-RE. The correct solution (John Boehner, Jan. 2010, Oct. 2010) showed up as the best answer with 66.94% voting correlation with Marshall in the specified time interval, a deviation of 1.94% from r_0. Other sensible solutions with similar relative strengths include (Michele Bachmann, Jan. 2009, Oct. 2010) with 3.77% deviation,

[15] http://data.bls.gov/timeseries/LNS14000000.
[16] http://www.dot.state.tx.us/txdot_library/drivers_vehicles/publications/crash_statistics/.
[17] http://www.govtrack.us/.
[18] http://en.wikipedia.org/wiki/NOMINATE_(scaling_method).

(Aaron Schock, Jan. 2009, Oct. 2010) with 1.62% deviation, and (Paul Ryan, Jan. 2009, Oct. 2010) with 4.32% deviation. The ordered answers provided by our algorithm thus significantly help reduce the difficulty of recovering missing parameter settings from hundreds of Representatives and various possible comparison periods.

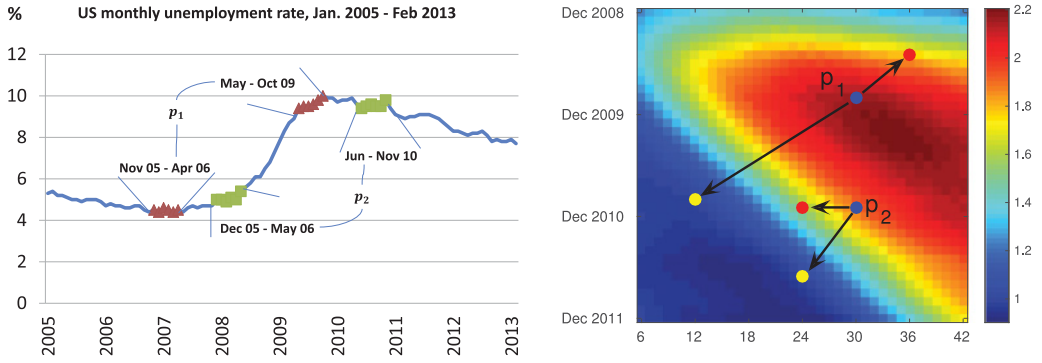
Next, given the reverse-engineered claim, we ran the CA-po algorithm for TSS-CA$_{ab}$ to find counterarguments with other sensible comparison periods that yield lower vote correlations between Marshall and Boehner. The top counterarguments, in decreasing sensibility, perturb the start of the period 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.

Finally, we ran the CA-po algorithm for TSS-CA$_u$ to find counterarguments comparing Marshall and other Representatives with equally high vote correlations but far from conservative (perturbing the target entity John Boehner, but compare the voting correlation over the same time interval). It turned out that we did not find Clyburn (which was used by factcheck.org), because our sensibility measure does not consider the roles played by the Representatives (Clyburn was the Democratic Whip). Instead, our algorithms found counterarguments involving Dennis Kucinich, Jim Oberstar, and Jackie Speier—all of whom not only vote more with Marshall, but are also more recognizable and liberal (according to the measures we used). More specifically, we use a triplet t_u to measure the quality of a counterargument by replacing Marshall with Representative e. The triplet consists of (i) u.rec (recognizability; the higher the better), (ii) u.ide (ideology; higher means more conservative, thus the lower the better), and (iii) vote correlation between u and Boehner (the higher the better). We have $t_{\text{Kucinich}} = (325707, -0.779, 41.79\%)$, $t_{\text{Oberstar}} = (135894, -0.533, 42.71\%)$, and $t_{\text{Speier}} = (160116, -0.48, 47.17\%)$, all of which dominate $t_{\text{Clyburn}} = (122309, -0.404, 41.29\%)$. Note that while higher recognizability is better most of the time, which direction is better for ideology and vote correlation depends on the purpose of the counterargument. In this example, a good counterargument would translate into "*Kucinich/Oberstar/Speier, a well-known Democrat, voted the same as Republican leader Boehner for as much as 41.79%/42.71%/47.17% between Jan. 2010 and Oct. 2010.*" While better measures of recognizability and ideology could find Clyburn as factcheck.org did, we believe that our counterarguments are also very strong. Other counterarguments we found automatically include Charles Rangel and Marcy Kaptur, for example.

6.1.3. Unemployment Claims. Many claims made by politicians are about trends in unemployment rate. In this experiment, we use UNEMP data to show that our framework finds high-quality counterarguments. Consider a subset of the data dated from January 2005 to February 2013 (Figure 8(a)). A WAC claim on this data can be stated in the following form: *The w-month average unemployment rate starting from time t increased/decreased by $|r - 1| \cdot 100\%$, compared to the w-month average starting d month before t.* Two of such claims are marked by red triangles and green squares, respectively, in Figure 8.

Claim 1. $p_1 = (w_1, t_1, d_1) = (6, \text{October 2009}, 30)$; $r_1 = 2.1487$ (114.87% increase).
Claim 2. $p_2 = (w_2, t_2, d_2) = (6, \text{November 2010}, 30)$; $r_2 = 1.8849$ (88.49% increase).

Both of the preceding claims point out increases in unemployment rate. A natural counterargument would point out a trend of decrease or lesser increase. We adopt the strength function $\mathsf{SR}(r; r_0) = r/r_0 - 1$ for $r_0 > 1$ from Section 2.1. For the sensibility function, we set $\sigma_w = 2.5$, $\sigma_t = 5$, and $\sigma_d = 10$. We use a four-level hierarchy for naturalness on t and d with $(\chi_1, \pi_1) = (e^0, 1)$ (month), $(\chi_2, \pi_2) = (e^1, 3)$ (quarter), $(\chi_3, \pi_3) = (e^2, 6)$ (half year), and $(e^3, 12)$ (year).



(a) US monthly unemployement rate, Jan. 2011 – Feb. 2013 (b) QRS

Fig. 8. Unemployment data and the query response surface for WAC claims. In (b), p_1 and p_2 correspond to the two WAC claims in (a), whose comparison intervals are represented by red triangles and green squares, respectively, both with 6-month windows.

Table II. Summary of Time Complexities of Algorithms

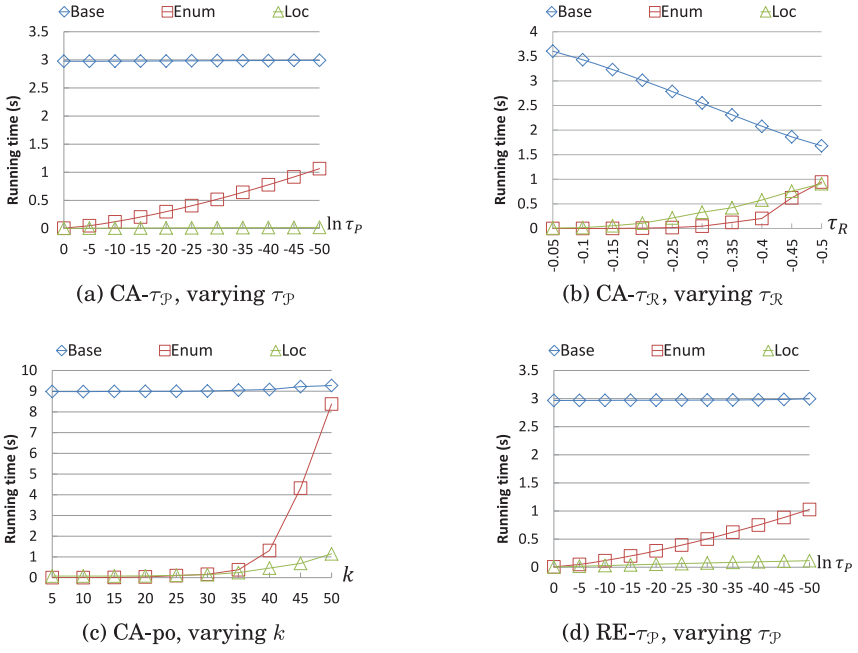
	Meta-problems	**WAC**	**TSS-CA$_{ab}$ TSS-RE**	**TSS-CA$_u$**
Base-$\tau_{\mathcal{P}}$	$O(\lvert\mathcal{P}\rvert(\mu_p+\mu_q))$	$O(n^3)$	$O(n^2)$	N/A
Base-$\tau_{\mathcal{R}}$	$O(\lvert\mathcal{P}\rvert(t\mu_p+\mu_q))$	$O(n^3)$	$O(n^2)$	$O(mn)$
Base-po	$O(\lvert\mathcal{P}\rvert(\mu_p \log t+\mu_q))$	$O(n^3 \log t)$	$O(n^2 \log t)$	$O(m(\log t+n))$
Enum-$\tau_{\mathcal{P}}$ Enum-po	$O(d\eta^d(\log \eta+\mu_q))$	$O(\tilde{r}^3 \log \tilde{r})$	$O(\tilde{r}^2 \log \tilde{r})$	N/A
Enum-$\tau_{\mathcal{R}}$	$O(d\eta^d(\log \eta+\mu_q))$	$O(\tilde{r}^3 \log \tilde{r})$	$O(\tilde{r}^2 \log \tilde{r})$	$O(m(\log t+n))$
Loc-$\tau_{\mathcal{P}}$	$O(\mu_d+m\mu_o)$	$O(\tilde{r}^2)$ - CA $O(\tilde{r}^2 \log \tilde{r})$ - RE	$O(\tilde{r} \log \tilde{r})$	N/A

From Figure 8(a), we see that Claim 1 reasonably captures the trend around its "current time" of October 2009, while for Claim 2 the unemployment rate starts to go down around its "current time" of November 2010. Intuitively, it should be easier to find high-sensibility counterarguments for Claim 2 than for Claim 1. Figure 8(b) visualizes the response surface (not relative strength surface) by fixing $w = 6$ and varying t and d. By setting $\tau_{\mathcal{R}} = 0$, we find the most sensible counterargument (solution to CA-$\tau_{\mathcal{R}}$) for Claim 2 to be to its left, a red dot pointed to by the horizontal arrow going out of p_2; that counterargument yields a relative strength of -17.62% with respect to r_2. To get an equally strong counterargument for Claim 1 (with a relative strength of -17.41% with respect to r_1), we need to go further to the upper-right of p_1. Furthermore, to find the most sensible counterargument that changes the trend of increase to a decrease, we need to perturb p_1 to as far as (6, October 2010, 12), while we only need to perturb p_2 to (6, July 2011, 24). These counterarguments are shown as yellow dots pointed to by arrows in Figure 8(b). The preceding results confirm our intuition that it is easier to find counterarguments for Claim 2, and imply that Claim 1 is more robust.

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 Loc (locus). Table II[19,20,21] summarizes the complexities of the meta-algorithms and their

[19]For TSS-RE, the complexity is for each of the m target entities.
[20]For Enum-$\tau_{\mathcal{R}}$, the complexity is for Enum$^{\text{partial}}\tau_{\mathcal{R}}$ for TSS-CA$_u$, and for Enum$^{\text{weak}}\tau_{\mathcal{R}}$ for the others.
[21]Time complexities of Loc-$\tau_{\mathcal{R}}$ and Loc-po are always factors of $O(\log \lvert\mathcal{P}\rvert)$ (i.e., $O(\log n)$) and $O(k \log \lvert\mathcal{P}\rvert)$ higher than their Loc-$\tau_{\mathcal{P}}$ counterparts; omitted for brevity.



(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 $\tau_{\mathcal{P}}$

Fig. 9. Running time of CA and RE algorithms for WAC claims on UNEMP.

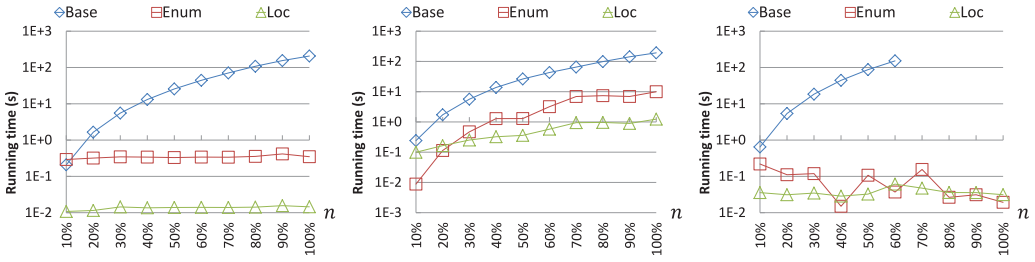
instantiations for WAC and TSS, as discussed in previous sections. In each cell of the table, the complexities are the same for both CA and RE problems, unless stated otherwise.

For brevity, when the context is clear, we shall use these names to refer to the respective algorithms for a given problem. 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.

Each data point in the figures below is obtained by averaging over 100 original claims with randomly generated parameter settings. Unless otherwise noted, 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 9(a) 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 Loc. However, as $\tau_{\mathcal{P}}$ decreases, the region of $\mathcal{P}$ meeting this threshold becomes larger. According to Equation (8), the interesting solution radius $\tilde{r}$ is determined by $\tau_{\mathcal{P}} - \tilde{r} = O(|\ln \tau_{\mathcal{P}}|^{1/2})$. Enum explores $O(|\ln \tau_{\mathcal{P}}|^{3/2})$ settings in $O(|\ln \tau_{\mathcal{P}}|^{3/2} \log |\ln \tau_{\mathcal{P}}|)$ time (Section 4.2.2), which explains Enum's super-linear increase in running time in Figure 9(a). Loc, with its powerful low-level building blocks, runs in time linear in $|\ln \tau_{\mathcal{P}}|$ ($O(\tilde{r}^2)$—Section 4.2.3). This trend is difficult to see in the figure, as Loc remains fast even with very low sensibility thresholds.

Varying $\tau_{\mathcal{R}}$ in CA-$\tau_{\mathcal{R}}$ for WAC on UNEMP. Figure 9(b) 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 (i.e., $\tilde{r}$ increases) and are harder for Enum and Loc to find. On the other hand, lower $\tau_{\mathcal{R}}$ makes Base faster because it needs to call



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

Fig. 10. Running time of CA algorithms for WAC claims on AUTO when varying data size.

SP with fewer parameter settings that meet the result strength threshold.[22] When $\tau_{\mathcal{R}}$ is as small as -0.5, a large portion of $\mathcal{P}$ must be examined by Enum and Loc. 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 Loc are faster than Base. Also, we see that for CA-$\tau_{\mathcal{R}}$, when $|\tau_{\mathcal{R}}|$ is as small as 0.4, Loc ($O(\tilde{r}^2 \log n)$—Sections 3.3, 4.2.3) holds no advantage over Enum ($O(\tilde{r}^3 \log \tilde{r})$—Section 4.2.2). Only when $\tau_{\mathcal{R}} < -0.4$ does the running time of Enum start to outgrow that of Loc.

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 9(c) shows the running time of the three algorithms for WAC claims on UNEMP when we vary k (effectively increasing $\tilde{r}$ as k increases). Enum and Loc show comparable performance up to $k = 35$. After that, the running time of Enum ($O(\tilde{r}^3 \log \tilde{r})$—Section 4.2.2) increases rapidly, and approaches that of Base ($O(n^3 \log t)$—Section 3.1). On the other hand, the running time of Loc ($O(k\tilde{r}^2 \log n$—Sections 3.3 and 4.2.3) 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. For RE-$\tau_{\mathcal{P}}$, the problem of reverse-engineering, Figure 9(d) compares the three algorithms for WAC claims on UNEMP. As we decrease the parameter sensibility threshold $\tau_{\mathcal{P}}$, we observe the same trend as in Figure 9(a) for CA-$\tau_{\mathcal{P}}$: Base is the slowest, while Loc ($O(|\ln \tau_{\mathcal{P}}| \log |\ln \tau_{\mathcal{P}}|)$—Section 4.2.3) scales better than Enum ($O(|\ln \tau_{\mathcal{P}}|^{3/2} \log |\ln \tau_{\mathcal{P}}|)$—Section 4.2.2) in the size of the high-sensibility region of the parameter space. Note that Loc for RE-$\tau_{\mathcal{P}}$ in Figure 9(d) is slightly slower than Loc for CA-$\tau_{\mathcal{P}}$ in Figure 9(a), because of the more expensive building block (OptP_0 vs. $\text{OptP}_{-\infty}$ 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 10, 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 10(b)), Enum shows a rate of increase in running time similar to Base, while Loc shows a slower rate of increase. This increasing trend is expected because more data points lead to more counterarguments with

[22] 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 Figure 9(a). In practice, when query evaluation is more expensive, the opposite may hold.

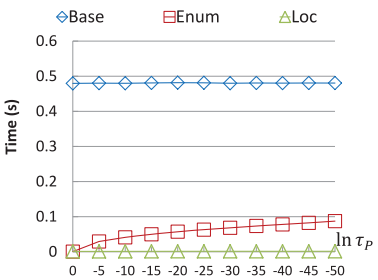


Fig. 11. Running time of CA-$\tau_{\mathcal{P}}$ algorithms for TSS-CA$_{ab}$ on VOTE, varying $\tau_{\mathcal{P}}$.

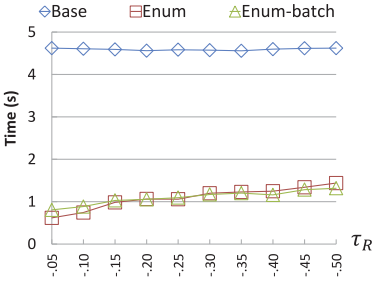


Fig. 12. Running time of CA-$\tau_{\mathcal{R}}$ algorithms for TSS-CA$_u$ on VOTE, varying $\tau_{\mathcal{R}}$.

required strength threshold. For CA-$\tau_{\mathcal{P}}$ (Figure 10(a)) and CA-po (Figure 10(c)), Base continues to suffer from bigger data sizes, but Enum and Loc remain fast. The reason is that Enum and Loc 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 Loc are orders of magnitude faster than Base.

Varying $\tau_{\mathcal{P}}$ in CA-$\tau_{\mathcal{P}}$ for TSS-CA$_{ab}$ on VOTE. We now turn to TSS claims on VOTE data. Figure 11 compares the three algorithms for TSS-CA$_{ab}$, that is, 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 9(a) and 9(d) for WAC claims: Loc performs best, while Base is the slowest by a big margin. The only notable difference is that the parameter space is 3D for WAC but only 2D here. Hence, Enum and Loc fare better here with an increasing search space, with time complexities $O(|\ln \tau_{\mathcal{P}}| \log |\ln \tau_{\mathcal{P}}|)$ ($O(\tilde{r}^2 \log \tilde{r})$—Section 5.2.2) and $O(|\ln \tau_{\mathcal{P}}|^{1/2} \log |\ln \tau_{\mathcal{P}}|)$ ($O(\tilde{r} \log \tilde{r})$—Section 5.2.3), respectively.

Varying $\tau_{\mathcal{R}}$ in CA-$\tau_{\mathcal{R}}$ for TSS-CA$_u$ on VOTE. Continuing with TSS claims on VOTE, we now compare algorithms for TSS-CA$_u$, that is, perturbing Representative u to be compared with the claim subject, while fixing the period of comparison, to find counterarguments involving liberal Representatives with high vote correlations. As discussed in Section 5.1, we consider the CA-$\tau_{\mathcal{R}}$ variant of the problem, which finds the counterarguments that meet the given result strength threshold $\tau_{\mathcal{R}}$ and are maximal with respect to the parameter sensibility relation $\preceq^{u_0}$ (no parameter sensibility function is defined in this case).

We compare only Base and Enum (more precisely, Enum$^{\text{partial}}$CA-$\tau_{\mathcal{R}}$), as no Loc algorithm is applicable here. For this experiment, a database system stores the full VOTE data, and we simply issue SQL queries to compute vote correlations; we do not store or preprocess VOTE data in memory. We observed that SQL queries dominate the running times. We also observed that using one SQL query to compute vote correlations for multiple Representatives in a batch is faster than using one query for each Representative. Therefore, for Base, we computed all vote correlations in a single SQL query; for Enum, we tested both batched (with five correlation computations per SQL query) and unbatched versions.

Figure 12 shows the results. As $|\tau_{\mathcal{R}}|$ increases, the number of parameter settings meeting the required result strength threshold decreases. Because Base evaluates the correlation between v and all possible u's using SQL queries (regardless of $\tau_{\mathcal{R}}$) and these SQL queries dominate the running time, the running time of Base remains roughly

constant. Enum simultaneously enumerates u's in descending order of $\mathsf{Nat}_{\mathsf{src}}$ and $\mathsf{Nat}_{\mathsf{src}}$, respectively, and tries to minimize the number of correlation computations. We see that Enum, with or without batching, outperforms Base by a big margin. Between the batched (shown as Enum-batch in Figure 12) and unbatched versions of Enum, we see that batching can be slower than no batching when $|\tau_{\mathcal{R}}|$ is small (where the total number of correlation computations needed is even lower than the batch size), but the advantage of batching starts to show when $|\tau_{\mathcal{R}}|$ is large.

7. DISCUSSION AND RELATED WORK

A large body of work on uncertain data management [Dalvi et al. 2009; Aggarwal 2009; Jampani et al. 2011] 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 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* [Ioannidis et al. 1992; Ganguly 1998; Hulgeri and Sudarshan 2003; D. et al. 2008], 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 [Yu et al. 2012]. Similar ideas have been used in understanding the sensitivity of such top-k rankings with respect to their parameter settings [Soliman et al. 2011; Mouratidis and Pang 2012]. Lin et al. [2013] uses a surface to concisely represent how the set of association rules varies with support and confidence settings.

Also related to the idea of retrieving more relevant results via query perturbation, is a large body of work on *query generalization and specialization* [Chaudhuri 1990]. Jensen and Snodgrass [1994] studied the generalization and specialization of relational queries along the temporal dimension. The problem has also been studied in the cooperative and hierarchical setting [Huh et al. 2000; Chu et al. 1991]. Beyond relational queries, the idea has also been applied to search queries [Koh et al. 2013].

The reverse-engineering problem is related to recent work on *query by output* [Tran et al. 2009] and *view synthesis* [Das Sarma et al. 2010], 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 [2010] consider how to modify a query such that it returns both the original result as well as additional desired tuples. He and Lo [2012] tackle the specific setting of linear preference top-k queries. Wu and Madden [2013] study how to use queries to explain away outliers in aggregate results. Roy and Suciu [2014] consider how removal of tuples by a predicate affects query results. While the work discussed previously is similar to our

problem in spirit, their search spaces and goals are very different, and none of them models query perturbations probabilistically.

As mentioned toward the end of Section 1, fact checking in general requires a repertoire of techniques including but not limited to the computational ones presented in this article, such as how to find datasets relevant to given claims and how to translate claims to queries. These questions are broadly related to areas including *Natural Language Processing (NLP)*, *Natural Language Querying (NLQ)* [Li et al. 2007, 2006; Popescu et al. 2003], *information integration* [Bernstein and Haas 2008; Doan et al. 2012], and *source selection* [Balakrishnan and Kambhampati 2011; Dong et al. 2009; Zhao et al. 2012]. Fully automatic solutions would be nice, but they are unlikely. Existing NLQ techniques cannot handle complex and/or ambiguously stated queries, which are common in our setting. Therefore, in general, we take the approach of combining automation with human input.

Fact checking with structured data has many applications in domains where assessments and decisions are increasingly driven by data. *Computational journalism* [Cohen et al. 2011a, 2011b] is one domain with the most pressing need for better fact-checking techniques. With the movement toward 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 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.

There are also claims that cannot be readily derived from structured data (e.g., "*does the health care reform create 'death panels'?*"). Recently, there has been interesting work that relies on *collective intelligence* [Quinn and Bederson 2011] for fact checking, either by involving experts or the public (e.g., factcheck.org, politifact.com, hypothes.is; see Giles [2012] for more discussion), or by collecting evidence from human-created data (e.g., Li et al. [2011], Yamamoto and Tanaka [2009], and Yamamoto et al. [2008]). This line of work is quite different from ours in style. Our quality measures have the advantage of objectivity that protects them from manipulation of and bias in personal opinions. Though far from a universal solution, our approach is attractive from claims based on structured data—which are increasingly common as more structured datasets become available either directly or by information extraction. Targeting this important type of tasks (or subtasks), our approach complements those that rely on human intelligence.

8. CONCLUSION AND FUTURE WORK

In this article, 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. For example, sampling and approximation are effective for large parameter spaces and data sizes. All algorithms in this article are exact; developing faster approximation algorithms would be a natural next step. Computing various claim quality measures

from the QRS, a subject that this article does not focus on, is clearly amenable to sampling. We are also working on extending the QRS framework to incorporate data changes as a dimension of perturbation orthogonal to parameter perturbations; this extension enables a new suite of useful and interesting questions related to fact checking. Specialized building blocks for many other claim types remain to be discovered. We have been working on a new parallel system [Walenz and Yang 2016] for perturbation analysis of database queries that cover new claim types, as well as the study of the orthogonal aspect of data perturbations. Another line of work is to go from fact checking to lead finding. Besides the interesting problems at the back-end, we are also working on making our techniques easier to apply. For parameters in the CA and RE problems, we have shown how the selection of thresholds $\tau_{\mathcal{R}}$ and $\tau_{\mathcal{P}}$ are supported interactively. For the claim modeling components in the problem definition, we rely on human experts to help define the parameterized query response function q and its domain, the relative strength function SR, and the parameterized relative sensibility SP. On top of that, selecting parameters for SP (e.g., covariance matrix of a Gaussian kernel), may involve a fair amount of human work, as the perception of a sensible perturbation is both user and context dependent. Along this line, we are investigating a learning-based approach that relies on user feedback to help choose appropriate parameters for the SP function.

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

Charu C. Aggarwal (Ed.). 2009. *Managing and Mining Uncertain Data*. Springer.

Raju Balakrishnan and Subbarao Kambhampati. 2011. SourceRank: Relevance and trust assessment for deep web sources based on inter-source agreement. In *Proceedings of the 2011 International Conference on World Wide Web*. 227–236.

Philip A. Bernstein and Laura M. Haas. 2008. Information integration in the enterprise. *Commun. ACM* 51, 9 (2008), 72–79.

Stephan Börzsönyi, Donald Kossmann, and Konrad Stocker. 2001. The skyline operator. In *Proceedings of the 2001 International Conference on Data Engineering*. 421–430.

Christian Buchta. 1989. On the average number of maxima in a set of vectors. *Inform. Process. Lett.* 33, 2 (1989), 63–65.

Surajit Chaudhuri. 1990. Generalization and a framework for query modification. In *Proceedings of the 6th International Conference on Data Engineering, 1990*. IEEE, 138–145.

Bernard Chazelle. 1988. A functional approach to data structures and its use in multidimensional searching. *SIAM J. Comput.* 17, 3 (1988), 427–462.

Wesley W. Chu, Qiming Chen, and Rei-Chi Lee. 1991. *Cooperative Query Answering via Type Abstraction Hierarchy*. Springer.

Sarah Cohen, James T. Hamilton, and Fred Turner. 2011a. Computational journalism. *Commun. ACM* 54, 10 (2011), 66–71.

Sarah Cohen, Chengkai Li, Jun Yang, and Cong Yu. 2011b. Computational journalism: A call to arms to database researchers. In *Proceedings of the 2011 Conference on Innovative Data Systems Research*.

Harish D., Pooja N. Darera, and Jayant R. Haritsa. 2008. Identifying robust plans through plan diagram reduction. In *Proceedings of the 2008 International Conference on Very Large Data Bases*. 1124–1140.

Nilesh N. Dalvi, Christopher Ré, and Dan Suciu. 2009. Probabilistic databases: Diamonds in the dirt. *Commun. ACM* 52, 7 (2009), 86–94.

Anish Das Sarma, Aditya G. Parameswaran, Hector Garcia-Molina, and Jennifer Widom. 2010. Synthesizing view definitions from data. In *Proceedings of the 2010 International Conference on Database Theory*. 89–103.

Mark De Berg, Marc Van Kreveld, Mark Overmars, and Otfried Cheong Schwarzkopf. 2000. *Computational Geometry*. Springer.

AnHai Doan, Alon Halevy, and Zachary Ives. 2012. *Principles of Data Integration* (1st ed.). Morgan Kaufmann.

Xin Luna Dong, Laure Berti-Equille, and Divesh Srivastava. 2009. Integrating conflicting data: The role of source dependence. *Proc. VLDB Endow.* 2, 1 (2009), 550–561.

Ronald Fagin, Amnon Lotem, and Moni Naor. 2003. Optimal aggregation algorithms for middleware. *J. Comput. System Sci.* 66, 4 (2003), 614–656.

Sumit Ganguly. 1998. Design and analysis of parametric query optimization algorithms. In *Proceedings of the 1998 International Conference on Very Large Data Bases*. 228–238.

Jim Giles. 2012. Truth goggles. *The New Scientist* 2882 (Sept. 2012), 44–47.

Jim Gray, Adam Bosworth, Andrew Layman, and Hamid Pirahesh. 1996. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-total. In *Proceedings of the 1996 International Conference on Data Engineering*. 152–159.

Dov Harel and Robert E. Tarjan. 1984. Fast algorithms for finding nearest common ancestors. *SIAM J. Comput.* 13, 2 (1984), 338–355.

Zhian He and Eric Lo. 2012. Answering why-not questions on top-k queries. In *Proceedings of the 2012 International Conference on Data Engineering*. 750–761.

Soon-Young Huh, Kae-Hyun Moon, and Hee-Seok Lee. 2000. A data abstraction approach for query relaxation. *Inf. Softw. Technol.* 42, 6 (2000), 407–418.

Arvind Hulgeri and S. Sudarshan. 2003. AniPQO: Almost non-intrusive parametric query optimization for nonlinear cost functions. In *Proceedings of the 2003 International Conference on Very Large Data Bases*. 766–777.

Yannis E. Ioannidis, Raymond T. Ng, Kyuseok Shim, and Timos K. Sellis. 1992. Parametric query optimization. In *Proceedings of the 1992 International Conference on Very Large Data Bases*. 103–114.

Ravi Jampani, Fei Xu, Mingxi Wu, Luis Leopoldo Perez, Chris Jermaine, and Peter J. Haas. 2011. The Monte Carlo database system: Stochastic analysis close to the data. *ACM Trans. Database Syst.* 36, 3 (2011), 18.

Christian S. Jensen and Richard Snodgrass. 1994. Temporal specialization and generalization. *IEEE Trans. Knowl. Data Eng.* 6, 6 (1994), 954–974.

Jia-Ling Koh, Kuang-Ting Chiang, and I.-Chih Chiu. 2013. The strategies for supporting query specialization and query generalization in social tagging systems. In *Database Systems for Advanced Applications*. Springer, 164–178.

Hsiang-Tsung Kung, Fabrizio Luccio, and Franco P. Preparata. 1975. On finding the maxima of a set of vectors. *J. ACM* 22, 4 (1975), 469–476.

Xian Li, Weiyi Meng, and Clement T. Yu. 2011. T.-verifier: Verifying truthfulness of fact statements. In *Proceedings of the 2011 International Conference on Data Engineering*. 63–74.

Yunyao Li, Ishan Chaudhuri, Huahai Yang, Satinder Singh, and H. V. Jagadish. 2007. DaNaLIX: A domain-adaptive natural language interface for querying XML. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*. 1165–1168.

Yunyao Li, Huahai Yang, and H. V. Jagadish. 2006. Constructing a generic natural language interface for an XML database. In *Proceedings of the 2006 International Conference on Extending Database Technology*. 737–754.

Xika Lin, Abhishek Mukherji, Elke A. Rundensteiner, Carolina Ruiz, and Matthew O. Ward. 2013. PARAS: A parameter space framework for online association mining. *Proc. VLDB Endow.* 6, 3 (2013), 193–204.

Kurt Mehlhorn and Stefan Näher. 1990. Dynamic fractional cascading. *Algorithmica* 5, 1–4 (1990), 215–241.

Kyriakos Mouratidis and HweeHwa Pang. 2012. Computing immutable regions for subspace top-k queries. *Proc.VLDB Endow.* 6, 2 (2012), 73–84.

Ana-Maria Popescu, Oren Etzioni, and Henry A. Kautz. 2003. Towards a theory of natural language interfaces to databases. In *Proceedings of the 2003 International Conference on Intelligent User Interfaces*. 149–157.

Alexander J. Quinn and Benjamin B. Bederson. 2011. Human computation: A survey and taxonomy of a growing field. In *Proceedings of the 2011 International Conference on Human Factors in Computing Systems*. 1403–1412.

Sudeepa Roy and Dan Suciu. 2014. A formal approach to finding explanations for database queries. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*. 1579–1590.

Mohamed A. Soliman, Ihab F. Ilyas, Davide Martinenghi, and Marco Tagliasacchi. 2011. Ranking with uncertain scoring functions: Semantics and sensitivity measures. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*. 805–816.

Robert Endre Tarjan. 1979. Applications of path compression on balanced trees. *J. ACM* 26, 4 (1979), 690–715.

Quoc Trung Tran and Chee-Yong Chan. 2010. How to ConQueR why-not questions. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*. 15–26.

Quoc Trung Tran, Chee-Yong Chan, and Srinivasan Parthasarathy. 2009. Query by output. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*. 535–548.

Brett Walenz and Jun Yang. 2016. Perturbation analysis of database queries. *Proc. VLDB Endow* 9, 14 (2016).

Eugene Wu and Samuel Madden. 2013. Scorpion: Explaining away outliers in aggregate queries. *Proc. VLDB Endow.* 6, 8 (June 2013), 553–564.

You Wu, Pankaj K. Agarwal, Chengkai Li, Jun Yang, and Cong Yu. 2012. On "one of the few" objects. In *Proceedings of the 2012 ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 1487–1495.

You Wu, Brett Walenz, Peggy Li, Andrew Shim, Emre Sonmez, Pankaj K. Agarwal, Chengkai Li, Jun Yang, and Cong Yu. 2014. iCheck: Computationally combating lies, d–ned lies, and statistics. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*. ACM, 1063–1066.

Yusuke Yamamoto and Katsumi Tanaka. 2009. Finding comparative facts and aspects for judging the credibility of uncertain facts. In *Proceedings of the 2009 International Conference on Web Information Systems Engineering*. 291–305.

Yusuke Yamamoto, Taro Tezuka, Adam Jatowt, and Katsumi Tanaka. 2008. Supporting judgment of fact trustworthiness considering temporal and sentimental aspects. In *Proceedings of the 2008 International Conference on Web Information Systems Engineering*. 206–220.

Albert Yu, Pankaj K. Agarwal, and Jun Yang. 2012. Processing a large number of continuous preference top-k queries. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*. 397–408.

Bo Zhao, Benjamin I. P. Rubinstein, Jim Gemmell, and Jiawei Han. 2012. A Bayesian approach to discovering truth from conflicting sources for data integration. *Proc. VLDB Endow.* 5, 6 (2012), 550–561.

Received June 2015; revised May 2016; accepted September 2016